CrossSection2dFEM 中声学模式的有限元分析及其在软件架构中的作用

基于 VocalTractLab3D 代码分析 (Blandin et al., 2022 [1])

2025 年 5 月 17 日

目录

A 矩形截面解析解公式 **1**

B 引言 **1**

C 软件架构概述 (基于 Blandin et al., 2022 [1]) **1**

D 理论基础：考虑曲率和面积变化的波动方程 (对应论文 Sec. II.B) **3**

D.1 几何变换 (对应论文 Sec. II.B.1) 3

D.2 变换后的波动方程 (对应论文 Sec. II.B.2) 3

D.3 一阶演化方程形式 4

E 多模态分解与耦合方程组 (对应论文 Sec. II.B.3) **4**

F 横向模式的定义：多模态分解的基础 **5**

G FEM 离散化 **6**

G.1 从 FEM 矩阵到多模态矩阵：投影关系 6

G.2 弱形式 7

G.3 基函数离散化 7

G.4 广义特征值问题 7

H `computeModes` 中的实现：与论文 [1] 的联系 **8**

H.1 矩阵组装 (对应论文 Eq. 32, 33) 8

H.2 求解特征问题 (对应论文 Eq. 31) 10

H.3 提取模式结果 (对应论文 Sec. II.C.1) 10

I 多模态矩阵的计算：为传播模型 [1] (Eq. (10)) 做准备 **11**

J 段间连接 (对应论文 Sec. II.C.2) **14**

K 轴向传播数值解法 (对应论文 Sec. II.C.3) **15**

L 边界条件与结果计算 (对应论文 Sec. II.D) **15**

M 声道几何表示与处理：从 CSV 导入到应用 **16**
M.1 引言 . . . 16
M.2 输入数据：CSV 文件格式与约定 . . . 16
M.2.1 文件结构 . . . 16
M.2.2 坐标系约定 . . . 17
M.2.3 缩放因子 $l(x)$ . . . 17
M.3 数据解析与初步处理 (`extractContoursFromCsvFile`) . . . 17
M.4 核心处理：几何变换与参数计算 (`createCrossSections`) . . . 18
M.4.1 几何变换：局部居中与全局补偿 . . . 18
M.4.2 其他计算 . . . 20
M.5 对象实例化与最终存储 (`addCrossSectionFEM` 与 `buildMesh`) . . . 21
M.6 应用：数据显示 (`SegmentsPicture` 与 `PropModesPicture`) . . . 21
M.6.1 横截面视图 (`PropModesPicture`) . . . 22
M.6.2 矢状面视图 (`SegmentsPicture`) . . . 22
M.7 工作流程概览 . . . 25
M.7.1 流程图 . . . 25
M.7.2 步骤总结 . . . 25

N 阅读指南与章节概览 **26**
N.1 十环节思维链 . . . 27
N.2 全文章节结构 . . . 27

O 几何切片与局部坐标（对应论文 Sec. II.A） **28**
O.1 CSV 字段 → 临时容器 . . . 28
O.2 局部 Z-轴居中 & 全局补偿 . . . 28
O.3 段长度与曲率参数 . . . 29
O.4 实例：截面 102 的变换前后 . . . 29

P 二维 FEM 横向模式与特征值求解（对应论文 Sec. II.B.2, App. V-A） **29**
P.1 弱式回顾 & 矩阵记号 . . . 29
P.2 网格密度选择与误差评估 . . . 30
P.3 特征求解器调用细节 . . . 30
P.4 模式后处理与符号约定 . . . 30
P.5 投影生成 C, D, E, K^{R2} . . . 30
P.6 单段验证脚本 . . . 30

Q 段内传播与 Magnus–Möbius 递推（对应论文 Sec. II.B.3, Eq. (48)–(52)） **31**
Q.1 传播矩阵 $\mathbf{M}(x)$ 拼装 . . . 31
Q.2 四阶 Magnus 展开实现 . . . 31
Q.3 阻抗与导纳的 Möbius 更新 . . . 31
Q.4 数值稳定性与步长选择 . . . 32

R 段间接口散射矩阵 F 及崩溃根因 **32**
R.1 **F** 的理论定义与代码生成 32
R.1.1 几何包含判定 32
R.2 接口阻抗/导纳递推再次回顾 33
R.3 崩溃复现与 gdb 栈 33
R.4 根因分析 33
R.5 修复思路概览 33

S 补丁实现与验证 **34**
S.1 代码改动总览 34
S.2 补丁 1：安全降级 34
S.3 补丁 2：接口使用端断言 34
S.4 编译与单元测试 34
S.5 长期方案：精确交集虚段 34
S.6 性能影响评估 35

A 矩形截面解析解公式

对于矩形截面，可以解析计算模式互耦矩阵。假设矩形尺寸为 $a \times b$，则可用以下公式计算多模态矩阵：

$$C_{mn} = \int_S \phi_m^* z \phi_n dS$$
$$D_{mn} = \int_S (\boldsymbol{\nabla}_\perp \phi_m)^* \cdot (z \boldsymbol{\nabla}_\perp \phi_n) dS$$
$$E_{mn} = \int_S \phi_m^* \mathbf{v} \cdot \boldsymbol{\nabla}_\perp \phi_n dS$$

此处省略完整公式，详见参考资料 [1] 附录。

B 引言

在 VocalTractLab3D 框架中，`CrossSection2dFEM` 类负责表示声道的任意二维横截面，并计算其声学特性。根据 Blandin et al. (2022 [1]) 的描述，该软件采用结合多模态方法和有限元 (FEM) 的策略。其中，**二维有限元方法 (2D FEM) 的核心作用是计算声道各个横截面的横向声学模式 (transverse modes, ϕ_n)**，这些模式随后被用于构建多模态传播模型，以模拟声波在考虑曲率和截面积变化的声道中的传播。

本文档的目的在于详细解析 `CrossSection2d.cpp` 中,特别是 `CrossSection2dFEM::computeModes` 函数内实现的 2D FEM 计算过程，阐明其如何根据论文 [1] 的理论计算出横向模式以及后续多模态分析所需的耦合矩阵，并说明其在整体软件架构中的位置。

C 软件架构概述 (基于 Blandin et al., 2022 [1])

VocalTractLab3D 的核心模拟架构基于 Blandin 等人 [1] 提出的高效 3D 声学模拟方法，其主要特点是结合了二维有限元（2D FEM）和多模态传播技术。其基本流程和架构可以概括为以下几

个关键阶段:

1. **几何处理与分段 (Geometry Processing & Segmentation)**
   - *输入:* 接受 3D 声道几何形状，可以是由 VocalTractLab 内置的发音模型生成，也可以是从外部文件（如 MRI 数据转换而来）导入。
   - *处理:* 将连续的 3D 声道沿中心线 (centerline) 切割成一系列短的**段 (segments)**。关键假设是：在每一段内部，其**横截面形状保持不变**，但允许其面积和位置（曲率）沿段的轴向变化。这种分段是应用多模态方法的基础 (论文 Sec. II.A, Fig. 1)。
   - *实现:* 这部分功能主要由 `VocalTract.cpp` 类及其依赖项处理，负责计算中心线、法线、截面轮廓等。
2. **横向模式计算 (Transverse Mode Computation via 2D FEM)**
   - *目的:* 对每个分段的代表性二维横截面，精确计算其横向声学模式 ϕ_n (特征函数) 和对应的特征值 γ_n^2（决定截止频率）。这些模式构成了后续声场分解的基础。
   - *方法:* 采用 **2D 有限元方法 (FEM)** 求解该横截面上的亥姆霍兹特征值问题（论文 Eq. 31），使用 Neumann 边界条件。
   - *实现:* **本文档后续章节将详细解析此步骤的实现**，位于 `CrossSection2dFEM::computeModes` 函数中，利用 CGAL 进行网格划分，利用 Eigen 求解 FEM 产生的广义特征值问题。
3. **计算多模态耦合矩阵 (Multimodal Coupling Matrices)**
   - *目的:* 基于 FEM 计算得到的模式 ϕ_n，预先计算一系列积分矩阵 (C, D, E, KR2, DR)。这些矩阵量化了模式之间由于声道几何（如曲率 κ、横截面坐标 z、边界类型 s）和物理效应（如壁面损耗 ζ_n）引起的耦合。
   - *方法:* 通过将 FEM 计算的中间矩阵（如质量矩阵、刚度矩阵、边界积分矩阵等）投影到模式基上获得（对应本文件 (15)-(19)）。
   - *实现:* 这部分计算同样在 `CrossSection2dFEM::computeModes` 中，紧随特征值求解之后进行。**本文档后续章节也将详细解析这些矩阵的计算**。
4. **计算段间连接/散射矩阵 (Junction/Scattering Matrices)**
   - *目的:* 描述声波在相邻两个具有不同截面形状或面积的段的界面处的散射和模式转换。
   - *方法:* 通过计算模式匹配积分得到模式转换矩阵 F (论文 Eq. 44)，并推导出阻抗/导纳在界面处的转换关系 (论文 Eq. 45-46)。论文提到，为了处理非包含截面的情况，可能引入零长度的交集截面段。
   - *实现:* 相关逻辑可能分布在 `Acoustic3dSimulation.cpp` (如 `computeJunctionMatrices` 或类似函数) 和 `CrossSection2dFEM.cpp` (存储和提供 F 矩阵)。
5. **轴向多模态传播 (Axial Multimodal Propagation)**
   - *目的:* 模拟声波（表示为各模式幅值向量 $\mathbf{p}$ 和辅助场 $\mathbf{q}$）沿分段声道轴向 x 的传播。
   - *方法:* 求解一个耦合的一阶常微分方程组（见本文件 (10)），其系数矩阵 $\mathbf{M}(x)$ 由之前计算的多模态矩阵 (C, D, E, KR2) 以及物理参数和几何参数（$k, l(x), l'(x), \kappa$ 等）构成。数值求解采用 Magnus-Möbius 格式（论文 Eq. 47-52），可以处理变化的截面积和曲率。

- *实现:* 这部分核心的传播计算逻辑位于 `Acoustic3dSimulation.cpp` (如 `propagateMagnus` 或 `propagateImpedAdmit` 函数) 和 `CrossSection2dFEM.cpp` (实现具体的传播步骤)。

6. **边界条件与激励 (Boundary Conditions & Excitation)**

- 在声门端（起始端）施加声源激励（如单位体积速度源）。
- 在口端（末端）施加辐射条件（如通过数值积分计算的辐射阻抗矩阵 ZoutNs, 论文 Eq. 53）或简化的边界条件（硬壁、零压力等）。
- 壁面损耗通过边界导纳系数 ζ 或 ζ_n 体现在传播方程中（见本文件 (18) 和 (7)）。
- *实现:* 边界条件的处理和声源的施加逻辑主要在 `Acoustic3dSimulation.cpp` 中。

7. **结果合成与输出 (Result Synthesis & Output)**

- 整个流程通常用于计算频率响应（传递函数 H(f), 论文 Eq. 56）或特定频率下的声场分布（论文 Eq. 54）。
- 通过组合不同频率的计算结果，可以合成声音信号。
- *实现:* `Acoustic3dSimulation.cpp` 负责协调计算并提供结果接口,GUI 部分(如 `Acoustic3dPage.cpp`) 或其他调用者获取这些结果进行显示或处理。

总结: 该架构的核心优势在于，通过 FEM 精确捕捉复杂横截面的模式特性，然后利用高效的多模态方法处理轴向传播，同时能够计入曲率和面积变化的影响，从而在计算效率和几何精度之间取得了良好的平衡，超越了传统的 TLM 和纯 FEM 方法。

D 理论基础：考虑曲率和面积变化的波动方程 (对应论文 Sec. II.B)

为了在多模态框架下处理弯曲且截面积变化的声道段，需要进行坐标变换并将波动方程转换到新的坐标系下。

D.1 几何变换 (对应论文 Sec. II.B.1)

引入一个从物理笛卡尔坐标 (X, Y, Z) 到一个局部坐标系 (x, y, z) 的变换。在这个局部坐标系中，声道段是直的且截面积恒定 (参考形状)。x 是沿着声道中心线的弧长坐标，y, z 是局部横截面坐标。中心线在 (X, Y, Z) 空间中的位置向量为 $\mathbf{s}(x)$，局部横截面基向量为 $\mathbf{n}_y, \mathbf{n}_z$ (构成 Frenet-Serret 标架的一部分)。引入一个缩放因子 $l(x)$ 来描述截面尺寸沿 x 的变化。一个物理点 $\mathbf{m}$ 的位置可以表示为:

$$\mathbf{m} = X\mathbf{i} + Y\mathbf{j} + Z\mathbf{k} = \mathbf{s}(x) + yl(x)\mathbf{n}_y + zl(x)\mathbf{n}_z. \tag{1}$$

其中 $\mathbf{t} = d\mathbf{s}/dx$ 是中心线的切向量，$\kappa(x)$ 是中心线的曲率。假设弯曲发生在 (X, Z) 平面内，即 $\mathbf{n}_y = \mathbf{j}$ 且 $\mathbf{n}_z$ 在 (X, Z) 平面内与 $\mathbf{t}$ 正交，则有 $d\mathbf{n}_y/dx = 0$ 和 $d\mathbf{n}_z/dx = -\kappa(x)\mathbf{t}$。

通过微分 Eq. (1) 并整理，可以得到两个坐标系之间的 Jacobian 矩阵 $\mathbf{J} = \frac{\partial(x,y,z)}{\partial(X,Y,Z)}$。其行列式为 $\det \mathbf{J} = 1/(fl^2)$，其中 $f(x, z) = 1 - z\kappa(x)l(x)$ 是一个与曲率相关的因子，$l' = dl/dx$。

D.2 变换后的波动方程 (对应论文 Sec. II.B.2)

原始的亥姆霍兹方程为 $(\Delta_X + k^2)p = 0$，其中 Δ_X 是笛卡尔坐标下的拉普拉斯算子，$k = \omega/c$ 是波数。利用坐标变换关系，可以将拉普拉斯算子变换到 (x, y, z) 坐标系下：

$$\Delta_X p = \det(\mathbf{J})\mathrm{div}(\mathbf{H}\boldsymbol{\nabla} p)$$

其中 $\boldsymbol{\nabla} = (\partial_x, \partial_y, \partial_z)^T$ 是新坐标系下的梯度算子，div 是对应的散度算子，而 $\mathbf{H}$ 是度量张量相关的矩阵：

$$\mathbf{H} = \frac{\mathbf{J}^T\mathbf{J}}{\det \mathbf{J}} = \frac{1}{f}\begin{pmatrix} l^2 & -yl'l & -zl'l \\ -yl'l & f^2 + (yl')^2 & yzl'^2 \\ -zl'l & yzl'^2 & f^2 + (zl')^2 \end{pmatrix}. \tag{2}$$

变换后的波动方程为：

$$\mathrm{div}(\mathbf{H}\boldsymbol{\nabla} p) + \frac{k^2}{\det \mathbf{J}} p = \mathrm{div}(\mathbf{H}\boldsymbol{\nabla} p) + k^2 f l^2 p = 0. \tag{3}$$

同样，壁面边界条件 $\boldsymbol{\nabla}_X p \cdot \mathbf{n}_X + jk\zeta p = 0$（其中 $\mathbf{n}_X$ 是物理外法线，$\zeta = Z_0/Z_w$ 是归一化壁面导纳）变换为：

$$\mathbf{H}\boldsymbol{\nabla} p \cdot \mathbf{n} + jk\zeta p = 0, \quad \text{for } y, z \in \Gamma_W, \tag{4}$$

其中 $\mathbf{n}$ 是 (x, y, z) 坐标系下的外法线，Γ_W 是管壁。

D.3 一阶演化方程形式

为了得到关于轴向坐标 x 的演化方程，引入辅助场 q，定义为声压梯度在 $\mathbf{H}$ 度量下的轴向分量：

$$q \equiv (\mathbf{H}\boldsymbol{\nabla} p) \cdot \mathbf{t}_{local} = \frac{l}{f}(l\partial_x p - yl'\partial_y p - zl'\partial_z p), \tag{5}$$

其中 $\mathbf{t}_{local} = (1, 0, 0)^T$ 是 (x, y, z) 坐标系下的轴向单位向量。将 Eq. (3) 展开并代入 q 的定义，可以得到关于 $(p, q)^T$ 的一阶偏微分方程组：

$$\partial_x \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} \frac{l'}{l}\mathbf{v} \cdot \boldsymbol{\nabla}_\perp & \frac{f}{l^2} \\ -\mathrm{div}_\perp(f\boldsymbol{\nabla}_\perp) - f(kl)^2 & \frac{l'}{l}\mathrm{div}_\perp(\mathbf{v}\cdot) \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix}, \tag{6}$$

其中 $\mathbf{v} = (y, z)^T$ 是横向坐标向量，$\boldsymbol{\nabla}_\perp = (\partial_y, \partial_z)^T$ 是横向梯度算子，$\mathrm{div}_\perp$ 是横向散度算子。相应的，变换后的边界条件 Eq. (4) 可以写为（假设 $\mathbf{n}$ 在边界上只有横向分量 $\mathbf{n}_\perp$，即 $n_x = 0$）：

$$f(\boldsymbol{\nabla}_\perp p) \cdot \mathbf{n}_\perp - \frac{l'}{l}(\mathbf{v} \cdot \mathbf{n}_\perp)q + jk\zeta p = 0. \tag{7}$$

Eq. (6) 和 Eq. (7) 是后续多模态分析的基础。

E 多模态分解与耦合方程组 (对应论文 Sec. II.B.3)

将声场 $p(x, y, z)$ 和辅助场 $q(x, y, z)$ 投影到一系列正交的二维横向模式 $\phi_n(y, z)$ 上。这些模式是对应横截面亥姆霍兹特征值问题的解（见下一节 FEM 部分）：

$$\begin{cases} p(x, y, z) = \sum_{n=0}^{\infty} p_n(x)\phi_n(y, z) = \mathbf{p}(x)^T\boldsymbol{\phi}(y, z) \\ q(x, y, z) = \sum_{n=0}^{\infty} q_n(x)\phi_n(y, z) = \mathbf{q}(x)^T\boldsymbol{\phi}(y, z) \end{cases} \tag{8}$$

其中 $\mathbf{p}(x) = [p_0(x), p_1(x), \dots]^T$ 和 $\mathbf{q}(x) = [q_0(x), q_1(x), \dots]^T$ 是待求的模式幅值向量，$\boldsymbol{\phi}(y,z) = [\phi_0, \phi_1, \dots]^T$ 是模式向量。横向模式满足正交归一化条件：

$$\begin{cases} \langle \phi_m, \phi_n \rangle = \int_S \phi_m^* \phi_n dS = \delta_{mn} \\ \langle \boldsymbol{\nabla}_\perp \phi_m, \boldsymbol{\nabla}_\perp \phi_n \rangle = \int_S (\boldsymbol{\nabla}_\perp \phi_m)^* \cdot (\boldsymbol{\nabla}_\perp \phi_n) dS = \gamma_m^2 \delta_{mn} \end{cases} \tag{9}$$

其中 γ_m^2 是模式 ϕ_m 对应的特征值。将 Eq. (8) 代入一阶 PDE 系统 Eq. (6)，然后用 ϕ_m^* 左乘并对横截面 S 积分，利用模式的正交性 Eq. (9) 以及边界条件 Eq. (7)（通过散度定理处理边界积分项，详见论文 Appendix V-B），可以得到关于模式幅值向量 $\mathbf{p}$ 和 $\mathbf{q}$ 的耦合常微分方程组 (ODE)：

$$\frac{d}{dx}\begin{pmatrix}\mathbf{p}\\ \mathbf{q}\end{pmatrix} = \mathbf{M}(x)\begin{pmatrix}\mathbf{p}\\ \mathbf{q}\end{pmatrix} = \begin{pmatrix}\mathbf{M}_1 & \mathbf{M}_2\\ \mathbf{M}_3 & \mathbf{M}_4\end{pmatrix}\begin{pmatrix}\mathbf{p}\\ \mathbf{q}\end{pmatrix} \tag{10}$$

其中 $\mathbf{M}(x)$ 是 $2N \times 2N$ 的传播矩阵（假设截断至 N 个模式），其四个子块矩阵为：

$$\mathbf{M}_1 = \frac{l'}{l}\mathbf{E} \tag{11}$$

$$\mathbf{M}_2 = \frac{1}{l^2}(\mathbf{I} - \kappa l \mathbf{C}) \tag{12}$$

$$\mathbf{M}_3 = \mathbf{K}^2 + \kappa l(\mathbf{C}(kl)^2 - \mathbf{D}) \tag{13}$$

$$\mathbf{M}_4 = -\frac{l'}{l}\mathbf{E}^T \tag{14}$$

这里的 $\mathbf{I}$ 是单位矩阵，而 $\mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{K}^2$ 是 $N \times N$ 的多模态耦合矩阵，其元素 (m,n) 定义为：

$$C_{mn} = \langle \phi_m z, \phi_n \rangle = \int_S \phi_m^* z \phi_n dS \tag{15}$$

$$D_{mn} = \langle \boldsymbol{\nabla}_\perp \phi_m, z\boldsymbol{\nabla}_\perp \phi_n \rangle = \int_S (\boldsymbol{\nabla}_\perp \phi_m)^* \cdot (z\boldsymbol{\nabla}_\perp \phi_n) dS \tag{16}$$

$$E_{mn} = \langle \phi_m \mathbf{v}, \boldsymbol{\nabla}_\perp \phi_n \rangle = \int_S \phi_m^* (y\partial_y \phi_n + z\partial_z \phi_n) dS \tag{17}$$

$$K_{mn}^2 = (\gamma_m^2 - (kl)^2)\delta_{mn} + jkl \sum_s \zeta_n^{(s)} K_{mn}^{R2,(s)} \tag{18}$$

$$K_{mn}^{R2,(s)} = \int_{\Gamma^{(s)}} \phi_m^* \phi_n d\Gamma \tag{19}$$

其中 Eq. (18) 中的求和 $\sum_s$ 遍历不同的边界类型（如果有的话），$\zeta_n^{(s)}$ 是模式 n 在类型为 s 的边界上的有效导纳（可以考虑粘滞热损耗），$K_{mn}^{R2,(s)}$ 是在类型 s 边界 $\Gamma^{(s)}$ 上的模式积分。这些耦合矩阵的物理意义：

- $\mathbf{C}$: 由曲率 κ 引起的、z 坐标相关的模式间耦合。
- $\mathbf{D}$: 由曲率 κ 引起的、与模式梯度相关的模式间耦合。
- $\mathbf{E}$: 由截面积变化率 l'/l 引起的模式间耦合。
- $\mathbf{K}^2$: 对角线项代表模式 m 的传播常数 $(\gamma_m^2 - (kl)^2)$，非对角线项（通过 K^{R2}）包含壁面损耗效应引起的模式间耦合。

求解耦合 ODE 系统 Eq. (10) 即可得到声波在声道段内的传播情况。

总结: 该架构的核心优势在于，通过 FEM 精确捕捉复杂横截面的模式特性，然后利用高效的多模态方法处理轴向传播，同时能够计入曲率和面积变化的影响，从而在计算效率和几何精度之间取得了良好的平衡，超越了传统的 TLM 和纯 FEM 方法。

F 横向模式的定义：多模态分解的基础

在上一节（多模态分解与耦合方程组）中，我们将三维声场 p 和辅助场 q 分解到了一系列二维横向模式 $\phi_n(y,z)$ 上（见 Eq. (8)）。这些模式 ϕ_n 及其对应的特征值 γ_n^2 是构建耦合常微分方程组 (Eq. (10)) 的核心要素。本节将明确定义这些模式所满足的物理方程。

根据论文 [1] (Sec. II.B 和 II.C.1)，这些模式 ϕ_n 在每个（参考形状的）横截面 Ω 上是二维亥姆霍兹特征值问题的解：

$$\nabla^2\phi_n + \gamma_n^2\phi_n = 0 \quad \text{在}\Omega \text{ 内}$$

其中 $\nabla^2 = \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ 是横向拉普拉斯算子。

边界条件选择为 Neumann 条件（零法向速度），这更接近声道壁的物理特性，从而确保了该方法更快的收敛性 [1] (Sec. II.C.1)：

$$\frac{\partial\phi_n}{\partial n} = 0 \quad \text{在}\partial\Omega \text{ 上}$$

其中 $\partial/\partial n$ 表示沿边界外法线方向的导数。

求解这个特征值问题即可得到所需的模式形状 ϕ_n (特征函数) 和特征值 γ_n^2 (与截止频率相关)。由于实际声道的横截面形状通常很复杂，无法直接求得解析解，因此需要采用数值方法。下一节（FEM 离散化）将详细介绍如何使用有限元方法 (FEM) 来求解这个问题。

G FEM 离散化

G.1 从 FEM 矩阵到多模态矩阵：投影关系

在详细介绍弱形式和离散化之前，我们先阐明有限元计算结果（模式特征向量）与最终所需的多模态耦合矩阵 ($\mathbf{C}$, $\mathbf{D}$, $\mathbf{E}$, $\mathbf{K}^{R2}$) 之间的关键联系。

FEM 的核心输出是横向模式 ϕ_n 及其对应的特征值 γ_n^2。在 FEM 中，模式 ϕ_n 并非直接得到解析表达式，而是通过一组节点值向量 $\boldsymbol{\xi}_n$ 来近似表示，该向量对应于所选的有限元基函数 e_i（例如，论文 [1] Eq. 30 中的线性拉格朗日单元）：

$$\phi_n(y,z) \approx \sum_{i=1}^{N} (\xi_n)_i e_i(y,z) = \boldsymbol{\xi}_n^T \mathbf{e}(y,z)$$

其中 N 是网格节点（或自由度）的数量，$(\xi_n)_i$ 是模式 n 在节点 i 上的幅值，$\mathbf{e}$ 是基函数向量。

另一方面，多模态传播模型（见 Eq. (10)）所需的耦合矩阵 $\mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{K}^{R2}$ 等是由涉及模式 ϕ_n 及其梯度的积分定义的（见 Eq. (15)-(19)）。例如：

$$C_{mn} = \langle \phi_m, z\phi_n \rangle = \int_S \phi_m^* z \phi_n dS$$

$$D_{mn} = \langle \boldsymbol{\nabla}_\perp\phi_m, z\boldsymbol{\nabla}_\perp\phi_n \rangle = \int_S (\boldsymbol{\nabla}_\perp\phi_m)^* \cdot (z\boldsymbol{\nabla}_\perp\phi_n) dS$$

将模式的 FEM 近似 $\phi_n = \sum_i (\xi_n)_i e_i$ 代入这些积分定义中，我们可以看到这些耦合矩阵可以通过涉及基函数 e_i 的积分来计算。以 $\mathbf{C}$ 矩阵为例（假设实数基函数和模式）：

$$C_{mn} = \int_S \left(\sum_i (\xi_m)_i e_i\right) z \left(\sum_j (\xi_n)_j e_j\right) dS = \sum_i \sum_j (\xi_m)_i (\xi_n)_j \left(\int_S e_i z e_j dS\right)$$

注意到括号中的积分 $\int_S e_i z e_j dS$ 正是论文 [1] (Eq. 38) 中定义的 **z 加权质量矩阵** $M_{ij}^Z = \langle ze_i, e_j \rangle$ 的元素。该矩阵（以及其他类似矩阵，如论文 [1] Eq. 32, 33, 39, 40, 41 中定义的 A_{ij}, M_{ij}, A_{ij}^Z, B_{ij},

$R_{ij}^{(s)}$)是在 FEM 组装阶段通过遍历网格单元计算得到的“中间”矩阵。代码中对应 `mass`, `stiffness`, `massY`, `stiffnessY`, `B`, `R[s]` 等。

因此，最终的多模态耦合矩阵 C_{mn} 可以通过将中间 FEM 矩阵 $\mathbf{M}^Z$ 投影到由 FEM 特征值求解器计算得到的模式特征向量 $\boldsymbol{\xi}_m$ 和 $\boldsymbol{\xi}_n$ 上来获得（对应论文 [1] Eq. 34）：

$$C_{mn} = \sum_i \sum_j (\xi_m)_i M_{ij}^Z (\xi_n)_j = \boldsymbol{\xi}_m^T \mathbf{M}^Z \boldsymbol{\xi}_n$$

类似地，其他耦合矩阵也可以通过将相应的中间 FEM 矩阵投影到模式特征向量上得到（对应论文 [1] Eq. 35-37）：

- $\mathbf{D}_{mn} = \boldsymbol{\xi}_m^T \mathbf{A}^Z \boldsymbol{\xi}_n$ (使用 z 加权刚度矩阵 $\mathbf{A}^Z$, 对应代码 `stiffnessY`)
- $\mathbf{E}_{mn} = \boldsymbol{\xi}_m^T \mathbf{B} \boldsymbol{\xi}_n$ (使用矩阵 $\mathbf{B}$, 对应代码 `B`)
- $\mathbf{K}_{mn}^{R2,(s)} = \boldsymbol{\xi}_m^T \mathbf{R}^{(s)} \boldsymbol{\xi}_n$ (使用边界质量矩阵 $\mathbf{R}^{(s)}$, 对应代码 `R[s]`)
- 论文未明确列出但代码中计算的 $\mathbf{D}_R^{(s)}$ 矩阵同样通过 $\mathbf{D}_{\mathrm{R}}^{(s)} = \boldsymbol{\xi}_m^T \mathbf{R}^{Y,(s)} \boldsymbol{\xi}_n$ 计算得到 (使用 z 加权边界质量矩阵 $\mathbf{R}^{Y,(s)}$, 对应代码 `RY[s]`)。

这里的 $\mathbf{M}^Z$, $\mathbf{A}^Z$, $\mathbf{B}$, $\mathbf{R}^{(s)}$, $\mathbf{R}^{Y,(s)}$ 等矩阵是标准的 FEM 组装产物，依赖于网格和基函数的选择。

总结来说，从 FEM 组装（得到 A, M, M^Z, A^Z, B, R, RY 等）到多模态耦合矩阵（C, D_N, E, K^{R2}, D_R 等）的关键步骤是：

1. **求解广义特征值问题 $A\boldsymbol{\xi}_n = \gamma_n^2 M \boldsymbol{\xi}_n$ 得到模式特征向量 $\boldsymbol{\xi}_n$ (对应代码 `m_modes`)。**
2. **将标准 FEM 组装得到的中间积分矩阵 (M^Z, A^Z, B, R, RY 等) 投影到这些模式特征向量 $\boldsymbol{\xi}_n$ 上，得到最终的多模态耦合矩阵 (对应代码 `m_C`, `m_DN`, `m_E`, `m_KR2`, `m_DR`)。**

这个过程在 `computeModes` 函数的后半部分实现，紧随特征值求解之后，如后文” 多模态矩阵计算” 小节所述。

G.2 弱形式

亥姆霍兹方程的弱形式为（如论文 [1] 中标准 FEM 流程所述）：寻找 $p \in H^1(\Omega)$ 使得对于所有测试函数 $v \in H^1(\Omega)$，满足：

$$\int_S \nabla_\perp p \cdot \nabla_\perp v \, dS = \gamma^2 \int_S pv \, dS \tag{20}$$

其中 $\nabla_\perp = (\frac{\partial}{\partial y}, \frac{\partial}{\partial z})^T$ 是横向梯度算子，积分在横截面 S (即 Ω) 上进行。

G.3 基函数离散化

使用 CGAL（`CrossSection2dFEM::buildMesh`）将区域 Ω 离散化为三角形网格。采用线性拉格朗日基函数（P1 单元，论文 Eq. 30 记为 e_i）近似模式形状 ϕ_n：

$$\phi_n(y,z) \approx \sum_{j=1}^{N} (\xi_n)_j e_j(y,z) \tag{21}$$

其中 $\boldsymbol{\xi}_n$ 是包含节点幅值的特征向量，$(xi_n)_j$ 是模式 n 在节点 j 的幅值，N 是节点总数。

G.4 广义特征值问题

现在我们将展示广义特征值问题是如何从弱形式和基函数离散化推导出来的。

首先，我们将基函数离散化形式 (Eq. (21)) 代入到亥姆霍兹方程的弱形式 (Eq. (41)) 中。对于一个特定的模式 ϕ_n (近似为 $\sum_j (\xi_n)_j e_j$) 和任意测试函数 v，弱形式变为：

$$\int_S \nabla_\perp \left(\sum_{j=1}^{N} (\xi_n)_j e_j \right) \cdot \nabla_\perp v \, dS = \gamma_n^2 \int_S \left(\sum_{j=1}^{N} (\xi_n)_j e_j \right) v \, dS$$

由于 $(\xi_n)_j$ 是与积分无关的常数系数，可以移出积分：

$$\sum_{j=1}^{N} (\xi_n)_j \left(\int_S \nabla_\perp e_j \cdot \nabla_\perp v \, dS \right) = \gamma_n^2 \sum_{j=1}^{N} (\xi_n)_j \left(\int_S e_j v \, dS \right)$$

接下来，应用**伽辽金方法 (Galerkin method)**。该方法的核心思想是选择测试函数 v 与基函数 e_i 来自同一个函数空间。具体来说，我们要求上式对所有的基函数 e_i ($i = 1, \ldots, N$) 都成立，即令 $v = e_i$：

$$\sum_{j=1}^{N} \left(\int_S \nabla_\perp e_j \cdot \nabla_\perp e_i \, dS \right) (\xi_n)_j = \gamma_n^2 \sum_{j=1}^{N} \left(\int_S e_j e_i \, dS \right) (\xi_n)_j \quad \text{对于所有} i = 1, \ldots, N$$

这个方程组构成了 $N \times N$ 的线性代数系统。我们观察到括号中的积分项正好对应于前面提到的**刚度矩阵** A 和**质量矩阵** M 的元素（考虑到矩阵的对称性 $A_{ij} = A_{ji}, M_{ij} = M_{ji}$）：

- 刚度矩阵元素：$A_{ij} = \int_S \nabla_\perp e_i \cdot \nabla_\perp e_j \, dS$ (对应论文 Eq. 32)
- 质量矩阵元素：$M_{ij} = \int_S e_i e_j \, dS$ (对应论文 Eq. 33)

将这些矩阵元素代入上述方程组，即可得到矩阵形式的**广义特征值问题**（对应论文 Eq. 31）：

$$A\boldsymbol{\xi}_n = \gamma_n^2 M \boldsymbol{\xi}_n \tag{22}$$

其中 $\boldsymbol{\xi}_n$ 是包含模式 n 节点幅值的特征向量，γ_n^2 是对应的特征值。

这里的矩阵 $\mathbf{A}$（刚度矩阵）和 $\mathbf{M}$（质量矩阵）是通过将弱形式 (Eq. (41)) 中的梯度项积分 ($\int \nabla_\perp p \cdot \nabla_\perp v$) 和函数本身积分 ($\int pv$) 在有限元基函数 e_i 上离散化得到的。求解这个在**二维横截面坐标系** (y, z) 下建立的广义特征值问题 (Eq. (22))，其物理意义在于找到该横截面在 Neumann 边界条件下**固有的横向声学振动模式（或称驻波模式）**。特征向量 $\boldsymbol{\xi}_n$ 给出了第 n 个模式在网格节点上的形状（振幅分布），而特征值 γ_n^2 则代表了该模式对应的**平方横向波数**，它直接决定了该模式的截止频率。

求解此广义特征值问题即可获得模式的节点表示 $\boldsymbol{\xi}_n$ 和特征值 γ_n^2，这是后续计算多模态耦合矩阵和进行传播模拟的基础。

H `computeModes` 中的实现：与论文 [1] 的联系

`CrossSection2dFEM::computeModes` 函数的核心任务是执行上述 FEM 计算，并为后续的多模态传播计算准备好所需的全部数据。

H.1 矩阵组装 (对应论文 Eq. 32, 33)

通过遍历网格单元，利用标准 FEM 技术和数值积分（3 点高斯）组装全局质量矩阵 `mass` (M) 和刚度矩阵 `stiffness` (A)。

代码片段 (CrossSection2d.cpp, computeModes 内):

```
// 初始化矩阵
numVert = (int)m_mesh.number_of_vertices();
mass = Matrix::Zero(numVert, numVert);
massY = Matrix::Zero(numVert, numVert);
stiffness = Matrix::Zero(numVert, numVert);
stiffnessY = Matrix::Zero(numVert, numVert);
B = Matrix::Zero(numVert, numVert);

// 循环遍历网格中的三角形单元
for (CDT::Finite_faces_iterator it = m_mesh.finite_faces_begin();
  it != m_mesh.finite_faces_end(); ++it)
{
  // 计算单元面积
  faceArea = 0.5 * abs(it->vertex(0)->point().x() *
    (it->vertex(1)->point().y() - it->vertex(2)->point().y())
    + it->vertex(1)->point().x() *
    (it->vertex(2)->point().y() - it->vertex(0)->point().y())
    + it->vertex(2)->point().x() *
    (it->vertex(0)->point().y() - it->vertex(1)->point().y()));

  // 计算雅可比矩阵等
  J[0][0] = 0.; J[0][1] = 0.; J[1][0] = 0.; J[1][1] = 0.;
  for (int p(0); p < 3; p++)
  {
    J[0][0] += (it->vertex(p)->point().x()) * dSdr[p];
    J[0][1] += (it->vertex(p)->point().y()) * dSdr[p];
    J[1][0] += (it->vertex(p)->point().x()) * dSds[p];
    J[1][1] += (it->vertex(p)->point().y()) * dSds[p];
  }
  detJ = J[0][0] * J[1][1] - J[0][1] * J[1][0];
  quadPtWeightDetJ = quadPtWeight * detJ / 2.;

  // 计算形函数导数
  for (int p(0); p < 3; p++)
  {
    dSdx[p] = (J[1][1] * dSdr[p] - J[0][1] * dSds[p]) / detJ;
    dSdy[p] = (J[0][0] * dSds[p] - J[1][0] * dSdr[p]) / detJ;
  }

  // 组装质量矩阵和刚度矩阵
  for (int j(0); j < 3; j++)
  {
    for (int k(0); k < 3; k++)
    {
      idxM = it->vertex(j)->info();
      idxN = it->vertex(k)->info();

      // 质量矩阵贡献
      mass(idxM, idxN) += (1. + (int)(j == k)) * faceArea / 12;

      // 刚度矩阵贡献
      stiffness(idxM, idxN) += ((
        it->vertex((j + 1) % 3)->point().y() -      // bm
```

```
            it->vertex((j + 2) % 3)->point().y()) *      // bm
            (it->vertex((k + 1) % 3)->point().y() -      // bn
            it->vertex((k + 2) % 3)->point().y()) +      // bn
            (it->vertex((j + 2) % 3)->point().x() -      // cm
            it->vertex((j + 1) % 3)->point().x()) *      // cm
            (it->vertex((k + 2) % 3)->point().x() -      // cn
            it->vertex((k + 1) % 3)->point().x())      // cn
            ) / faceArea / 4;
        }
      }
    }
```

Listing 1: 组装质量矩阵和刚度矩阵

H.2 求解特征问题 (对应论文 Eq. 31)

使用 `Eigen::GeneralizedSelfAdjointEigenSolver` 求解 $A\boldsymbol{\xi}_n = \gamma_n^2 M\boldsymbol{\xi}_n$。

代码片段 (CrossSection2d.cpp, computeModes 内):

```
// 求解广义特征值问题
Eigen::GeneralizedSelfAdjointEigenSolver<Matrix> eigenSolver(stiffness, mass);
```

Listing 2: 求解广义特征值问题

H.3 提取模式结果 (对应论文 Sec. II.C.1)

- **特征值 γ_n^2 和截止频率** f_{cn}: 从求解器提取特征值，计算截止频率，存入 `m_eigenFreqs`。这些 γ_n^2 值直接用于后续多模态传播方程中的 K^2 项（见本文件 (18)）。
- **特征向量 $\boldsymbol{\xi}_n$ (模式形状节点值)**: 从求解器提取特征向量，存入 `m_modes`。这些向量定义了模式 ϕ_n，是计算所有多模态耦合矩阵的基础。
- **多模态传播方程**: 这些模式和特征值被代入 Eq. (6) 和 Eq. (7)，构成后续多模态传播计算的基础。

代码片段 (CrossSection2d.cpp, computeModes 内):

```
m_eigenFreqs.clear();
// 根据最大截止频率确定需要的模式数量
idx = 0;
maxWaveNumber = pow(2 * M_PI * simuParams.maxCutOnFreq / simuParams.sndSpeed, 2);
while ((eigenSolver.eigenvalues()[idx] < maxWaveNumber) && (idx < eigenSolver.eigenvalues().size()
    ))
{
  // 计算截止频率并存储
  m_eigenFreqs.push_back(sqrt(eigenSolver.eigenvalues()[idx])
                         * simuParams.sndSpeed / 2. / M_PI);
  idx++;
}
m_modesNumber = idx;

// 平面波模式频率强制设为0
m_eigenFreqs[0] = 0.;

// 提取特征向量 (模式形状)
```

```
  m_modes = eigenSolver.eigenvectors().block(0, 0, numVert, m_modesNumber);

  // 获取模式的最大和最小振幅
  m_maxAmplitude.clear();
  m_minAmplitude.clear();
  m_maxAmplitude.reserve(m_modesNumber);
  m_minAmplitude.reserve(m_modesNumber);

  // 确定第一个模式的符号，以使所有模式具有一致的符号约定
  if (eigenSolver.eigenvectors().col(0)[0] > 0)
  {
    signFirstMode = 1.;
  }
  else
  {
    signFirstMode = -1.;
  }

  // 应用符号约定并计算振幅范围
  for (int m(0); m < m_modesNumber; m++)
  {
    // 所有模式乘以第一个模式的符号
    m_modes.col(m) *= signFirstMode;
    // 获取模式的最大振幅
    m_maxAmplitude.push_back(m_modes.col(m).maxCoeff());
    // 获取模式的最小振幅
    m_minAmplitude.push_back(m_modes.col(m).minCoeff());
  }
```

Listing 3: 提取特征值和特征向量

I 多模态矩阵的计算：为传播模型 [1] (Eq. (10)) 做准备

FEM 计算模式后，`computeModes` 函数继续计算一系列额外的矩阵。这些矩阵是构建多模态传播方程（Eq. (10)）所需的耦合矩阵 $\mathbf{C}, \mathbf{D}, \mathbf{E}, \mathbf{K}^2$ 的基础。

计算流程紧密依照论文 [1] Section II.C.1 和 Appendix V-A 的推导：

首先，通过 FEM 组装几个基于单元形函数 e_i 的中间积分矩阵（使用高斯积分）。这些矩阵对应于耦合矩阵定义 (Eq. (15)-(19)) 中积分内的项，在 FEM 基上的表示：

- `mass` $\rightarrow M_{ij} = \langle e_i, e_j \rangle$ (质量矩阵, 论文 Eq. 33)
- `stiffness` $\rightarrow A_{ij} = \langle \boldsymbol{\nabla}_\perp e_i, \boldsymbol{\nabla}_\perp e_j \rangle$ (刚度矩阵, 论文 Eq. 32)
- `massY` $\rightarrow M^Z_{ij} = \langle z e_i, e_j \rangle$ (z 加权质量矩阵, 论文 Eq. 38, 用于计算 $\mathbf{C}$。注意这里的 z 加权源于论文 [1] 对弯曲平面（含全局 Z 轴）的假设，详见第 3.1 节)
- `stiffnessY` $\rightarrow A^Z_{ij} = \langle z\boldsymbol{\nabla}_\perp e_i, \boldsymbol{\nabla}_\perp e_j \rangle$ (z 加权刚度矩阵, 论文 Eq. 39, 用于计算 $\mathbf{D}$。同样，z 加权与弯曲平面的假设有关，见第 3.1 节)
- `B` $\rightarrow B_{ij} = \langle \mathbf{v} e_i, \boldsymbol{\nabla}_\perp e_j \rangle$ (论文 Eq. 40, 用于计算 $\mathbf{E}$)
- `R[s]` $\rightarrow R^{(s)}_{ij} = \int_{\Gamma^{(s)}} e_i e_j d\Gamma$ (边界质量矩阵, 论文 Eq. 41, 用于计算 $\mathbf{K}^{R2,(s)}$)
- `RY[s]` $\rightarrow R^{Y,(s)}_{ij}$ - (z 加权边界质量矩阵, 代码中有，论文未明确列出，但用于计算 $\mathbf{D}^{(s)}_R$)

其中 $\mathbf{v} = (y, z)^T$。代码中的 `massY` 和 `stiffnessY` 使用 Y 命名，但从公式看对应的是 z 坐标加权。代码中的 'DN' 矩阵对应论文中的 D 矩阵。

代码片段 *(CrossSection2d.cpp, computeModes 内，组装中间矩阵):*

```
// 高斯积分点和权重设置
double quadPtCoord[3][2]{ {1. / 6., 1. / 6.}, {2. / 3., 1. / 6.}, {1. / 6., 2. / 3.} };
double quadPtWeight = 1. / 3.;
double S[3][3];
for (int i(0); i < 3; i++)
{
  S[i][0] = 1. - quadPtCoord[i][0] - quadPtCoord[i][1];
  S[i][1] = quadPtCoord[i][0];
  S[i][2] = quadPtCoord[i][1];
}

// 遍历三角形面单元
for (CDT::Finite_faces_iterator it = m_mesh.finite_faces_begin();
  it != m_mesh.finite_faces_end(); ++it)
{
  // ... 计算雅可比矩阵、行列式等 ...

  // 计算单元的质量矩阵和刚度矩阵与位置加权的版本
  for (int j(0); j < 3; j++)
  {
    for (int k(0); k < 3; k++)
    {
      idxM = it->vertex(j)->info();
      idxN = it->vertex(k)->info();

      // 循环高斯积分点
      for (int q(0); q < 3; q++)
      {
        // 计算 massY (z加权质量矩阵)
        massY(idxM, idxN) += Yrs[q] * S[q][j] * S[q][k] * quadPtWeightDetJ;

        // 计算 stiffnessY (z加权刚度矩阵)
        stiffnessY(idxM, idxN) += Yrs[q] * (dSdx[j] * dSdx[k] + dSdy[j] * dSdy[k]) *
            quadPtWeightDetJ;

        // 计算 B 矩阵 (用于E矩阵的中间矩阵)
        B(idxM, idxN) += (Xrs[q] * S[q][j] * dSdx[k] + Yrs[q] * S[q][j] * dSdy[k]) *
            quadPtWeightDetJ;
      }
    }
  }
}

// 计算边界积分矩阵 R 和 RY
R.clear();
RY.clear();
// 为每个不同的边界类型创建一个R和RY矩阵
for (int s(0); s < m_surfIdxList.size(); s++)
{
  R.push_back(Matrix::Zero(numVert, numVert));
  RY.push_back(Matrix::Zero(numVert, numVert));
}

// 遍历边界段
for (int s(0); s < m_meshContourSeg.size(); s++)
```

```
54  {
55    // ... 确定边界段对应的表面类型 ...
56
57    // 计算线段长度
58    segLength = sqrt(pow(m_points[m_meshContourSeg[s][0]][0] -
59      m_points[m_meshContourSeg[s][1]][0], 2) +
60      pow(m_points[m_meshContourSeg[s][0]][1] -
61        m_points[m_meshContourSeg[s][1]][1], 2));
62
63    // 组装边界积分矩阵
64    for (int j(0); j < 2; j++)
65    {
66      for (int k(0); k < 2; k++)
67      {
68        idxM = m_meshContourSeg[s][j];
69        idxN = m_meshContourSeg[s][k];
70
71        // R矩阵贡献
72        R[idx](idxM, idxN) += (1. + (double)(j == k)) * segLength / 6.;
73
74        // RY矩阵贡献 (z加权边界积分)
75        RY[idx](idxM, idxN) += segLength / 12.; // y-加权项
76      }
77    }
78  }
```

Listing 4: 组装用于多模态矩阵的中间积分矩阵

然后，通过将这些中间矩阵投影到 FEM 计算得到的模式特征向量 $\boldsymbol{\xi}_n$ (即 $\phi_n = \sum_i (\xi_n)_i e_i$) 上，获得最终的多模态矩阵（论文 Eq. 34-37）：

- $\mathbf{C}_{mn} = \boldsymbol{\xi}_m^T \mathbf{M}^Z \boldsymbol{\xi}_n$ (对应代码 `m_C`)
- $\mathbf{D}_{mn} = \boldsymbol{\xi}_m^T \mathbf{A}^Z \boldsymbol{\xi}_n$ (对应代码 `m_DN`)
- $\mathbf{E}_{mn} = \boldsymbol{\xi}_m^T \mathbf{B} \boldsymbol{\xi}_n$ (对应代码 `m_E`)
- $\mathbf{K}_{mn}^{R2,(s)} = \boldsymbol{\xi}_m^T \mathbf{R}^{(s)} \boldsymbol{\xi}_n$ (对应代码 `m_KR2[s]`)
- $\mathbf{D}_{\mathrm{R}}^{(s)} = \boldsymbol{\xi}_m^T \mathbf{R}^{Y,(s)} \boldsymbol{\xi}_n$ (对应代码 `m_DR[s]`)

注意，论文中的 $\mathbf{K}^2$ 矩阵 (Eq. (18)) 是由特征值 γ_m^2、波数 k、缩放因子 l、边界导纳 $\zeta_n^{(s)}$ 以及这里计算出的 $\mathbf{K}^{R2,(s)}$ 组合而成，通常在传播计算时实时构建。

代码片段 *(CrossSection2d.cpp, computeModes 内，计算最终矩阵)*:

```
1   // 初始化多模态矩阵
2   m_C = Matrix::Zero(m_modesNumber, m_modesNumber);
3   m_DN = Matrix::Zero(m_modesNumber, m_modesNumber);
4   m_E = Matrix::Zero(m_modesNumber, m_modesNumber);
5
6   // 计算 C, DN, 和 E 矩阵
7   for (int m(0); m < m_modesNumber; m++)
8   {
9     for (int n(0); n < m_modesNumber; n++)
10    {
11      // 计算 C: \xi_m^T * massY * \xi_n
12      m_C(m, n) = (m_modes.col(m)).transpose() * massY * m_modes.col(n);
13
```

```
      // 计算 D_N: \xi_m^T * stiffnessY * \xi_n
      m_DN(m,n) = (m_modes.col(m)).transpose() * stiffnessY * m_modes.col(n);

      // 计算 E: \xi_m^T * B * \xi_n
      m_E(m,n) = (m_modes.col(m)).transpose() * B * m_modes.col(n);
    }
  }

  // 计算 DR 和 KR2 (按表面类型)
  m_DR.clear();
  m_KR2.clear();

  for (int s(0); s < m_surfIdxList.size(); s++)
  {
    m_DR.push_back(Matrix::Zero(m_modesNumber, m_modesNumber));
    m_KR2.push_back(Matrix::Zero(m_modesNumber, m_modesNumber));

    for (int m(0); m < m_modesNumber; m++)
    {
      for (int n(0); n < m_modesNumber; n++)
      {
        // 计算 D_R^(s): \xi_m^T * RY[s] * \xi_n
        m_DR.back()(m,n) = (m_modes.col(m)).transpose() * RY[s] * m_modes.col(n);

        // 计算 K_{R^2}^(s): \xi_m^T * R[s] * \xi_n
        m_KR2.back()(m,n) = (m_modes.col(m)).transpose() * R[s] * m_modes.col(n);
      }
    }
  }
```

Listing 5: 计算最终的多模态矩阵

这些矩阵 C, D, E, K_{R^2} 正是构成论文 [1] 中多模态传播矩阵 $\mathbf{M}$ (Eq. (11)-(14)) 的核心部分。**通过 2D FEM 精确计算这些与具体横截面形状相关的矩阵，再代入多模态传播方程 (Eq. (10))，是论文 [1] 提出的结合 FEM 和多模态方法以实现高效且精确 3D 模拟的关键**。论文 [1] (Fig. 3) 通过与解析解对比验证了这些矩阵计算的准确性。

J 段间连接 (对应论文 Sec. II.C.2)

当声波从一个声道段 (记为 a) 传播到另一个具有不同横截面形状或面积的段 (记为 b) 时，需要在交界面处满足声学连续性条件。假设段 a 的出口截面 S_a 与段 b 的入口截面 S_b 相接。

基于声压 p 和法向体积速度相关的量 q 在公共界面 $S_{common} = S_a \cap S_b$ 上的连续性（对于非公共部分，通常假设 $q = 0$），可以推导出两段模式幅值向量之间的关系。

令 $\mathbf{p}_a, \mathbf{q}_a$ 为段 a 出口处的模式幅值，$\mathbf{p}_b, \mathbf{q}_b$ 为段 b 入口处的模式幅值。如果 $S_a \subseteq S_b$，则连续性条件（详见论文 Appendix V-C）导出：

$$\mathbf{p}_a = \frac{l_b}{l_a}\mathbf{F}\mathbf{p}_b \tag{23}$$

$$\mathbf{q}_b = \frac{l_b}{l_a}\mathbf{F}^T\mathbf{q}_a \tag{24}$$

其中 l_a, l_b 分别是两段在交界面处的缩放因子，$\mathbf{F}$ 是模式匹配矩阵 (见本文件 (25))，其元素为：

$$F_{mn} = \int_{S_a} \phi_m^{(a)*}\phi_n^{(b)} dS \tag{25}$$

注意论文 Eq. 44 的形式是在笛卡尔坐标下积分，这里给出了在模式定义所在的 (y,z) 坐标下的等效形式（假设 l_a, l_b 在积分面上恒定）。这个矩阵描述了模式从一个基到另一个基的投影。

利用声阻抗 $\boldsymbol{\mathcal{Z}}$ ($p = \mathcal{Z}q$) 和声导纳 $\boldsymbol{\mathcal{Y}}$ ($q = \mathcal{Y}p$) 的定义，可以得到阻抗和导纳在界面处的传递关系：

$$\boldsymbol{\mathcal{Z}}_a = \left(\frac{l_b}{l_a}\right)^2 \mathbf{F}\boldsymbol{\mathcal{Z}}_b\mathbf{F}^T \tag{26}$$

$$\boldsymbol{\mathcal{Y}}_b = \left(\frac{l_b}{l_a}\right)^2 \mathbf{F}^T\boldsymbol{\mathcal{Y}}_a\mathbf{F} \tag{27}$$

对于 S_a 不包含于 S_b 的一般情况，论文 [1] 提到可以引入一个零长度、截面为 $S_a \cap S_b$ 的虚拟段来处理。

K 轴向传播数值解法 (对应论文 Sec. II.C.3)

耦合 ODE 系统 Eq. (10) 通常没有解析解，需要数值求解。一种高效且稳定的方法是基于 Riccati 方程和 Magnus-Möbius 格式。

首先，将 ODE 系统写成矩阵形式 $\frac{d\mathbf{w}}{dx} = \mathbf{M}(x)\mathbf{w}$，其中 $\mathbf{w} = (\mathbf{p}^T, \mathbf{q}^T)^T$。从位置 x_n 到 x_{n+1} 的解可以通过传递矩阵 联系：

$$\begin{pmatrix}\mathbf{p}(x_{n+1})\\ \mathbf{q}(x_{n+1})\end{pmatrix} = (x_{n+1}, x_n)\begin{pmatrix}\mathbf{p}(x_n)\\ \mathbf{q}(x_n)\end{pmatrix} = \begin{pmatrix}{}_1 & {}_2\\ {}_3 & {}_4\end{pmatrix}\begin{pmatrix}\mathbf{p}(x_n)\\ \mathbf{q}(x_n)\end{pmatrix} \tag{28}$$

Magnus 展开提供了计算 的近似方法。论文 [1] 使用了四阶 Magnus 格式：

$$(x_{n+1}, x_n) \approx \exp\left(\frac{\Delta x}{2}(\mathbf{M}(x_a) + \mathbf{M}(x_b)) + \frac{\sqrt{3}(\Delta x)^2}{12}[\mathbf{M}(x_b), \mathbf{M}(x_a)]\right) \tag{29}$$

其中 $\Delta x = x_{n+1} - x_n$，$[\mathbf{A}, \mathbf{B}] = \mathbf{AB} - \mathbf{BA}$ 是矩阵换位子，积分点 x_a, x_b 为：

$$x_a = x_n + (\frac{1}{2} - \frac{\sqrt{3}}{6})\Delta x \tag{30}$$

$$x_b = x_n + (\frac{1}{2} + \frac{\sqrt{3}}{6})\Delta x \tag{31}$$

计算矩阵指数 $\exp(\cdot)$ 是主要计算量。

为了避免直接传播可能指数增长的 $\mathbf{p}, \mathbf{q}$，通常传播阻抗 $\boldsymbol{\mathcal{Z}}$ 或导纳 $\boldsymbol{\mathcal{Y}}$ 矩阵。利用传递矩阵 的分块形式，可以得到阻抗和导纳的 Möbius 变换传播关系：

$$\boldsymbol{\mathcal{Z}}(x_{n+1}) = ({}_1\boldsymbol{\mathcal{Z}}(x_n) + {}_2)({}_3\boldsymbol{\mathcal{Z}}(x_n) + {}_4)^{-1} \tag{32}$$

$$\boldsymbol{\mathcal{Y}}(x_{n+1}) = ({}_3 + {}_4\boldsymbol{\mathcal{Y}}(x_n))({}_1 + {}_2\boldsymbol{\mathcal{Y}}(x_n))^{-1} \tag{33}$$

传播通常从口端向声门端计算阻抗/导纳，然后再从声门端向口端计算声压/速度。

L 边界条件与结果计算 (对应论文 Sec. II.D)

整个模拟流程需要合适的边界条件和结果提取方法。

- **声门端 (Glottis, $x = 0$):** 通常施加一个已知的声源条件，例如给定体积速度源。这通常通过设置入射波的模式幅值或直接设置 $q_{in}(0)$ 来实现。例如，对于均匀体积速度 Q_{in}，平面波模式的 $q_{in,0} = -j\omega\rho Q_{in}/S_0$（这里 q 可能与论文定义差一个因子）。得到入口阻抗 $\boldsymbol{\mathcal{Z}}_{in}(0)$ 后，可计算 $\mathbf{p}_{in}(0) = \boldsymbol{\mathcal{Z}}_{in}(0)\mathbf{q}_{in}(0)$。

- **口端 (Mouth, $x = L$):** 施加辐射边界条件。这通常表示为一个辐射阻抗矩阵 $\boldsymbol{\mathcal{Z}}_{out}(L)$，它将口端的模式声压 $\mathbf{p}_{out}(L)$ 与模式体积速度 $\mathbf{q}_{out}(L)$ (或相关量) 联系起来。该矩阵可以通过数值积分计算，例如假设口端在一个无限大障板中（论文 Eq. 53）：

$$\mathcal{Z}_{out,mn}(L) = -\frac{j\omega\rho}{2\pi}\int_S\int_{S'}\phi_m^*(y,z)\frac{e^{-jkR}}{R}\phi_n(y',z')dS'dS$$

  其中 $R = \sqrt{(y-y')^2+(z-z')^2}$ 是口面两点间距离，S 是口端横截面。这个矩阵 $\boldsymbol{\mathcal{Z}}_{out}(L)$ 作为从口端向声门端传播阻抗的初始条件。

- **结果计算:**
  - **传递函数 H(f):** 计算声门输入（如单位体积速度）到某个输出点（如口端中心压力或辐射场某点压力）的频率响应。论文 Eq. 56 定义为：

$$H(f) = \frac{P_o(f)}{A_g G(f)}$$

    其中 $P_o(f)$ 是输出点声压，A_g 是声门面积，$G(f)$ 是输入的均匀速度谱。
  - **声场分布:** 通过计算得到的模式幅值 $\mathbf{p}(x)$ 和 $\mathbf{q}(x)$，可以利用 Eq. (8) 重建任意位置 (x,y,z) 的声压 p。辐射声场可以通过口端的模式分布，利用 Rayleigh-Sommerfeld 积分计算（论文 Eq. 54）：

$$p_{rad}(\mathbf{r}) = \int_S \frac{j\omega\rho}{2\pi}v_n(y',z')\frac{e^{-jk|\mathbf{r}-\mathbf{r}'|}}{|\mathbf{r}-\mathbf{r}'|}dS'$$

    其中 v_n 是口端的法向速度，需要从 $\mathbf{p}(L)$ 和 $\mathbf{q}(L)$ 计算得到。

M 声道几何表示与处理：从 CSV 导入到应用

M.1 引言

本节旨在详细阐述 VTL3D 系统如何处理声道几何信息，重点关注从用户提供的 CSV 文件导入数据开始，经过一系列解析、关键几何变换、参数计算，最终生成可用于声学模拟和 GUI 显示的内部表示的全过程。系统的核心在于将离散的二维横截面信息及其空间关系转化为一系列 `CrossSection2dFEM` 对象 (类定义位于 `CrossSection2d.h`)，这些对象封装了每个声道段的几何形状、有限元网格以及其他相关物理参数。

整个流程的核心数据容器是 `Acoustic3dSimulation` 类 (定义位于 `Acoustic3dSimulation.h`) 中的 `m_crossSections` 成员变量 (一个 `std::vector<std::unique_ptr<CrossSection2d>>`)，它存储了代表整个处理后声道的所有段对象。关键的 C++ 函数包括负责数据解析的 `extractContoursFromCsvFile` (位于 `Acoustic3dSimulation.cpp:4949`)，执行核心几何变换与参数计算的 `createCrossSections` (位于 `Acoustic3dSimulation.cpp:5121`)，用于实例化和存储对象的 `addCrossSectionFEM` (位于 `Acoustic3dSimulation.cpp:531`)，生成网格的 `buildMesh` (位于 `CrossSection2d.cpp:321`)，以及在 GUI 中绘制矢状图时计算几何顶点坐标的 `getSegmentPts` (位于 `SegmentsPicture.cpp:574`)。

特别地，本节将详细分析在 `createCrossSections` 函数中执行的几何变换——局部轮廓 Z 轴居中和全局中心点补偿调整。这些变换基于特定的数学公式（见 (34) 和 (35)），对于确保局部坐标系与全局坐标系的一致性至关重要，是应用 Blandin 等人论文 [1] 中描述的 3D 坐标变换理论以进行精确声学模拟的基础。

M.2 输入数据：CSV 文件格式与约定

M.2.1 文件结构

根据代码分析 (`Acoustic3dSimulation::extractContoursFromCsvFile` 函数，位于 `Acoustic3dSimulation.` 和示例文件 (`contour_s1_a.csv`)，CSV 文件格式定义如下：

- 文件由成对的行组成，每对行代表一个声道横截面及其在全局坐标系中的位置和方向。
- **奇数行**: 包含全局 X 坐标和局部 Y 坐标相关数据：
  1. 中心点全局 X 坐标 (`double`)
  2. 法线全局 X 分量 (`double`)
  3. 起始端缩放因子 (`double`, $l(x_0)$ 的近似值)
  4. 轮廓点局部 Y 坐标序列 (`double`...)
- **偶数行**: 包含全局 Y 坐标和局部 Z 坐标相关数据：
  1. 中心点全局 Y 坐标 (`double`)
  2. 法线全局 Y 分量 (`double`)
  3. 末端缩放因子 (`double`, $l(x_1)$ 的近似值)
  4. 轮廓点局部 Z 坐标序列 (`double`...)
- 字段分隔符为分号 (';')。

M.2.2 坐标系约定

理解坐标系是正确解读 CSV 数据的关键：

- **全局坐标系 (X, Y):** CSV 文件中的中心点坐标 (每对行的第一个字段) 和法线向量 (第二个字段) 是在 VTL3D 的全局二维坐标系（通常对应于矢状面视图）中定义的。
- **局部坐标系 (y, z):** CSV 文件中第四个字段开始的轮廓点坐标序列，是定义在每个横截面自身的**局部二维坐标系**中的。奇数行提供的序列对应局部 **y** 坐标，偶数行提供的序列对应局部 **z** 坐标。这个局部坐标系的原点（在几何变换之前）是该截面的中心点，其轴向由法线向量确定。

M.2.3 缩放因子 $l(x)$

CSV 文件中每对行的第三个字段分别定义了该声道段起始端 ($l_{in,i}$) 和末端 ($l_{out,i}$) 的缩放因子。这个因子 $l(x)$ 描述了截面尺寸（面积）如何沿着段的中心线 x 变化。即使局部轮廓形状（由局部 y, z 坐标定义）在段内保持不变，$l(x)$ 的变化也会导致物理横截面积的变化。代码内部会使用 $l_{in,i}$ 和 $l_{out,i}$ 来插值计算段内任意位置的 $l(x)$ 及其导数 $l'(x)$。这些值是构建考虑面积变化的波动方程和多模态传播矩阵 $\mathbf{M}(x)$ 的关键输入。同时，在绘制矢状图时，它们也用来调整每个段端面的宽度，以视觉上反映面积变化。

M.3 数据解析与初步处理 (extractContoursFromCsvFile)

当用户选择导入 CSV 文件后，extractContoursFromCsvFile 函数 (位于 Acoustic3dSimulation.cpp:4949) 被调用，其主要职责是：

1. **解析基础信息：** 逐对读取 CSV 行，解析出全局中心点 $\mathbf{s}_i = (X_i, Y_i)$、全局法线 $\mathbf{N}_i = (N_{x,i}, N_{y,i})$ 和缩放因子 $l_{in,i}, l_{out,i}$。法线向量在此阶段被归一化，得到单位法线 $\hat{\mathbf{n}}_i$。这些信息被暂存到临时的 std::vector 容器中（如 centerLine, normals, scalingFactors）。
2. **构建初始局部轮廓：** 解析每对行中剩余的局部坐标点序列（局部 y, z），并使用这些点按顺序构建一个 CGAL 的 Polygon_2 对象，代表该截面在局部坐标系下的原始轮廓 $P_i = \{\mathbf{p}_{local,k} = (y_{local,k}, z_{local,k})\}$。相关的表面索引信息（如果存在）也会被提取。构建的轮廓多边形被存储在临时向量 contours 中。
3. **轮廓简化 (可选)：** 如果用户设置了简化选项 (通过 simplifyContours 参数)，则使用 CGAL 的 Polyline_simplification_2::simplify 算法对构建的 Polygon_2 对象进行顶点简化。
4. **输出临时数据：** 函数执行完毕后，输出一系列包含原始几何数据和定位信息的临时向量（主要是 contours, surfaceIdx, centerLine, normals, scalingFactors），这些数据将作为下一步核心处理的输入。

M.4 核心处理：几何变换与参数计算 (createCrossSections)

在获取了初步解析的数据后，createCrossSections 函数 (位于 Acoustic3dSimulation.cpp:5121) 承担起进行关键几何处理和计算的任务。其中最重要的一步是执行几何变换，以确保后续声学计算的准确性。

M.4.1 几何变换：局部居中与全局补偿

a. 动机与前提： 为了能够正确应用论文 [1] 中的 3D 坐标变换理论，必须确保每个截面的局部坐标系 (y, z) 与其在全局坐标系 (X, Y) 中的位置和方向保持一致。这要求对从 CSV 直接解析出的原始数据进行调整。在 VTL3D 中，这个调整过程通常在参数 m_simuParams.curved 设置为 true 时执行，主要包含以下两个核心步骤，由特定数学公式定义：

b. 步骤 1：局部轮廓 Z 轴居中： 首先，需要将每个截面的局部轮廓 P_i 沿着其局部 z 轴进行平移，使其几何中心位于 z=0 处。这通过计算轮廓在 z 轴上的中心 z_c 并应用平移向量 $\Delta\mathbf{p}_{local} = (0, -z_c)$ 来实现。

- 计算 Z 中心：

$$z_{min} = \min_k(z_{local,k})$$
$$z_{max} = \max_k(z_{local,k})$$
$$z_c = \frac{z_{min} + z_{max}}{2}$$

- 应用平移变换得到居中后的轮廓点 $\mathbf{p}'_{local,k}$：

$$\mathbf{p}'_{local,k} = \mathbf{p}_{local,k} + \Delta\mathbf{p}_{local} = (y_{local,k}, z_{local,k} - z_c) \tag{34}$$

代码对应 (Eq. (34))： 下列代码片段展示了如何计算平移向量 `shiftVec` (基于 $z_c = (z_{min} + z_{max})/2$) 并将其应用于轮廓顶点 `contours[i][j]` (位于 `Acoustic3dSimulation.cpp` 约 5185-5189 行)：

```
// 计算 z 轴中心 (z_c = (bboxes.back()[2] + bboxes.back()[3]) / 2.)
shiftVec = Vector(0., -(bboxes.back()[2] + bboxes.back()[3]) / 2.);
// 创建平移变换
Transformation translate(CGAL::TRANSLATION, shiftVec);
// 应用变换到当前截面的所有轮廓顶点
for (int j(0); j < contours[i].size(); j++)
{
  contours[i][j] = transform(translate, contours[i][j]);
}
```

Listing 6: 局部轮廓 Z 轴居中代码实现

结果是得到 z 轴居中的新局部轮廓 $P'_i = \{\mathbf{p}'_{local,k}\}$。

c. 步骤 2：全局中心线点位置补偿调整： 仅调整局部轮廓是不够的，必须相应地调整全局中心点 $\mathbf{s}_i$，以补偿局部坐标系的平移。调整的依据是上一步计算出的局部 Z 中心 z_c 和该截面的全局单位法线 $\hat{\mathbf{n}}_i$。调整后的全局中心点 $\mathbf{s}'_i$ 由下式给出：

$$\mathbf{s}'_i = \mathbf{s}_i + z_c \hat{\mathbf{n}}_i \tag{35}$$

代码对应 (Eq. (35))： 紧随局部轮廓居中之后，代码通过以下方式调整全局中心点 `centerLine[i]` 以补偿平移（位于 `Acoustic3dSimulation.cpp` 约 5190-5191 行）：

```
// centerLine[i] = centerLine[i] + z_c * normals[i]
// (注意: shiftVec.y() = -z_c, normals[i] 是单位法线)
centerLine[i].x -= shiftVec.y() * normals[i].x;
centerLine[i].y -= shiftVec.y() * normals[i].y;
```

Listing 7: 全局中心点补偿调整代码实现

经过这两个步骤，调整后的中心线轨迹 $\mathbf{s}'(x)$ 与经过公式 (34) 处理的局部轮廓坐标系 (y, z) 在几何上保持了一致性。

d. 示例：非居中矩形 （示例内容保持不变，因为它清晰地展示了公式 (34) 和 (35) 的应用）假设 CSV 文件包含以下一对行，定义了一个横截面：

```
2.0;0.707;1.1;1;1;-1;-1
3.0;0.707;1.2;1.5;0.5;0.5;1.5
```

1. 初始数据解析：

- 全局中心点: $\mathbf{s} = (2.0, 3.0)$
- 全局法线: $\hat{\mathbf{n}} \approx (0.7071, 0.7071)$ (归一化后)
- 缩放因子: $l_0 = 1.1, l_1 = 1.2$
- 原始局部轮廓点 P: $(1, 1.5), (1, 0.5), (-1, 0.5), (-1, 1.5)$ (局部 y, z)

2. 局部轮廓沿 Z 轴居中 (步骤 1):

- $z_{min} = 0.5$, $z_{max} = 1.5$
- $z_c = (0.5 + 1.5)/2 = 1.0$
- 根据公式 (34)，新的局部轮廓点 P': $(1, 0.5), (1, -0.5), (-1, -0.5), (-1, 0.5)$ (居中)

3. 全局中心线点补偿调整 (步骤 2):

- 根据公式 (35),调整后的中心点 $\mathbf{s}' = \mathbf{s} + z_c\hat{\mathbf{n}} \approx (2.0, 3.0) + 1.0 \times (0.7071, 0.7071) = (2.7071, 3.7071)$

最终，得到的居中局部轮廓 P' 和调整后的全局中心点 $\mathbf{s}'$ 被用于后续处理。

M.4.2 其他计算

除了核心的几何变换外，`createCrossSections` 函数还会利用解析和变换后的数据计算其他重要参数，**特别是描述相邻截面间几何关系的段长度和曲率参数**:

- **截面面积与边界框:** 计算每个截面的总面积 (通过调用 `Polygon_2::area()` 方法累加) 并存储在 `totAreas` 中；计算每个截面的局部边界框 (通过 `Polygon_2::bbox()` 方法) 并存储在 `bboxes` 中，同时更新整个几何体的最大边界框 `m_maxCSBoundingBox`。
- **段长度 (`length`):** 对于连接截面 $i-1$ 和 i 的段，其长度 L_i 直接通过计算两个（可能经过调整的）中心点 $\mathbf{s}'_{i-1}$ 和 $\mathbf{s}'_i$ 之间的欧氏距离得到（代码位于 `Acoustic3dSimulation.cpp` 约 5400 行):

$$L_i = ||\mathbf{s}'_i - \mathbf{s}'_{i-1}|| \simeq \texttt{centerLine[i-1].getDistanceFrom(centerLine[i])}$$

- **曲率参数 (`prevCurvRadius`, `prevAngle`):** 为了描述中心线的弯曲程度以及如何从一个段过渡到下一个段，代码计算了两个关键的曲率参数。这些参数是针对**上一个**截面（例如，在处理段 i 时，计算的是围绕截面 $i-1$ 的参数）定义的，通过调用辅助函数 `getCurvatureAngleShift` (位于 `Acoustic3dSimulation.cpp:4849`) 来获得。该函数接收两个相邻截面的中心点 $(\mathbf{s}_{i-1}, \mathbf{s}_i)$ 和单位法线 $(\hat{\mathbf{n}}_{i-1}, \hat{\mathbf{n}}_i)$ 作为输入，计算得出:
  - **曲率半径 (`radius` / `prevCurvRadius`)**: R_{i-1} 代表拟合中心点 $\mathbf{s}_{i-1}$ 并与两条法线相切（近似）的圆弧半径。其计算基于两条法线的交点以及该交点到中心点的距离相关的几何关系:

$$R_{i-1} = \frac{(\mathbf{s}_i - \mathbf{s}_{i-1}) \times \hat{\mathbf{n}}_i}{\hat{\mathbf{n}}_i \times \hat{\mathbf{n}}_{i-1}} \qquad \text{(二维叉乘)}$$

代码对应 (曲率半径): (位于 `Acoustic3dSimulation.cpp` 约 4869-4870 行)

```
// 计算与 P1 相关的"伪半径"
radii[0] = ((P2.y - P1.y) * N2.x - (P2.x - P1.x) * N2.y) /
           (N2.x * N1.y - N2.y * N1.x);
// 直接使用 radii[0] 作为曲率半径
radius = radii[0];
```

  - **曲率角度 (`angle` / `prevAngle`)**: α_{i-1} 代表从法线 $\hat{\mathbf{n}}_{i-1}$ 到法线 $\hat{\mathbf{n}}_i$ 的旋转角度。通过计算两条法线的反正切值 (`atan2`) 并取差值，再标准化到 $(-\pi, \pi]$ 区间得到:

$$\alpha_{i-1} = \text{normalizeAngle}(\text{atan2}(N_{y,i}, N_{x,i}) - \text{atan2}(N_{y,i-1}, N_{x,i-1}))$$

代码对应 (曲率角度): (位于 Acoustic3dSimulation.cpp 约 4878-4889 行)

```
// 计算两个法线的角度
angles[0] = fmod(atan2(N1.y, N1.x) + 2. * M_PI, 2. * M_PI);
angles[1] = fmod(atan2(N2.y, N2.x) + 2. * M_PI, 2. * M_PI);
// 计算直接角度差
angle = angles[1] - angles[0];
// 标准化到 (-pi, pi]
if ((2. * M_PI - abs(angle)) < abs(angle))
{
  if (signbit(angle))
  {
    angle = 2. * M_PI - abs(angle);
  }
  else
  {
    angle = abs(angle) - 2. * M_PI;
  }
}
```

这些计算得到的参数（如 prevCurvRadius, prevAngle）随后与段长度 L_i 一起，在调用 addCrossSectionFEM 时传递给 CrossSection2dFEM 对象，用于后续计算出口中心点 (ctrLinePtOut) 等。

- 如果 m_simuParams.curved 为 false，则执行将中心线拉直并统一法线的操作（此时曲率参数通常为 0）。

M.5 对象实例化与最终存储 (addCrossSectionFEM 与 buildMesh)

在 createCrossSections 函数完成对所有截面数据的处理（包括几何变换和参数计算）后，它会遍历这些处理后的数据，并为每个声道段调用 addCrossSectionFEM 函数 (位于 Acoustic3dSimulation.cpp:5

1. **传递调整后数据:** 调用 addCrossSectionFEM 时，传递的关键参数是经过几何变换后的数据：**调整后的入口中心点** $\mathbf{s}'_i$ (来自 centerLine[i-1]，由公式 (35) 计算得到)，入口单位法线 $\hat{\mathbf{n}}_i$ (来自 normals[i-1])，以及 Z 轴居中后的局部轮廓 P'_{i-1} (来自 contours[i-1]，由公式 (34) 处理得到)。此外，还传递了计算出的面积、段长度 L_i (length)、曲率参数（角度 α_i，半径 R_i），以及从 CSV 或计算得到的缩放因子 $l_{0,i}, l_{1,i}$ (存储在 prevScalingFactors 数组中传递给构造函数)。

2. **对象创建:** addCrossSectionFEM 内部通过 new CrossSection2dFEM(...) 创建一个新的 CrossSection2dFEM 对象，并将上述处理后的几何信息和参数存储在该对象内部。特别是，**调整后的入口中心点 $\mathbf{s}'_i$ 被存储起来，将作为后续 ctrLinePtIn() 方法的返回值**。

3. **存储到核心容器:** 创建的对象通过 std::unique_ptr<CrossSection2d> 管理，并被添加到 Acoustic3dSimulation 类的核心成员变量 **m_crossSections** 向量中。这个向量最终包含了代表整个声道的所有段对象。

4. **网格生成:** 在 CrossSection2dFEM 对象被创建并添加到 m_crossSections 后（通常在其构造函数或后续的初始化调用中），会调用其 buildMesh() 方法 (位于 CrossSection2d.cpp:321)。此方法使用内部存储的、经过 Z 轴居中处理的轮廓 m_contour (即 P') 和 CGAL 库来生成该截面的二维有限元网格 m_mesh (类型为 CDT)。

至此，所有从 CSV 导入并经过核心处理（特别是几何变换）后的声道段几何形状、空间定位（包括调整后的入口中心点 $\mathbf{s}'_i$ 和计算出口点所需的参数）以及 FEM 网格信息都已准备就绪，并统一存储在 `Acoustic3dSimulation::m_crossSections` 中，等待后续的声学计算或 GUI 显示调用。

M.6 应用：数据显示 (`SegmentsPicture` 与 `PropModesPicture`)

存储在 `Acoustic3dSimulation` 实例 (`m_simu3d`) 的 `m_crossSections` 中的数据是 GUI 显示声道几何的基础。

M.6.1 横截面视图 (`PropModesPicture`)

此视图（通常在 GUI 右上角）显示选定单个横截面的细节：

- 获取当前选中的活动段索引 `m_activeSegment`。
- 调用 `m_simu3d->crossSection(m_activeSegment)` 从 `m_crossSections` 向量中获取对应的 `CrossSection2dFEM` 对象指针 `sec`。
- 根据用户的显示选择（轮廓、网格、模式等），调用相应的方法绘制。例如，调用 `sec->contour()` 会绘制存储在 `m_contour` 中的、经过 Z 轴居中处理的局部轮廓 (P')；调用 `sec->triangulation()` 则绘制生成的 FEM 网格 `m_mesh`。

为了更直观地理解 Z 轴居中（在 11.4.1 节中描述）的效果，图 1 展示了由 Python 模拟脚本生成的、针对 'contour_s1_a.csv' 文件中第 102 个截面的横截面视图。该图清晰地显示了原始轮廓（浅红色虚线）及其 Z 轴几何中心（紫色星号），以及经过平移变换后的居中轮廓（深蓝色实线）如何以局部坐标原点 (0,0) 为其新的 Z 轴中心。正是这个居中后的轮廓 (P') 被用于后续的网格生成和声学计算。

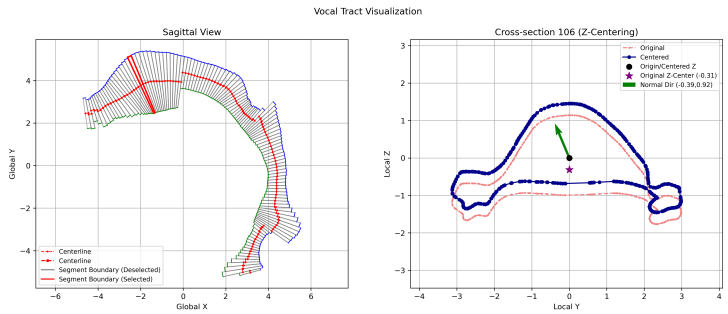


图 1: 截面 102 的 Z 轴居中效果示意图 (由 Python 模拟脚本生成)。图中显示了原始局部轮廓（浅红色虚线）、原始 Z 轴中心（紫色星号）、居中后的局部轮廓（深蓝色实线）、局部坐标原点/居中后的 Z 轴中心（黑色点）以及该截面的入口法线方向（绿色箭头）。

M.6.2 矢状面视图 (`SegmentsPicture`)

此视图（通常在 GUI 左上角）旨在展示整个声道的二维轮廓和中心线，由 `SegmentsPicture::draw` 方法 (位于 `SegmentsPicture.cpp:107`) 负责绘制。

- 遍历所有段：通过 `for` 循环遍历 `m_simu3d->numberOfSegments()`，并在循环内调用 `sec = m_simu3d->crossSection(i)` 从 `m_crossSections` 获取第 i 个段对象。

- **中心线绘制:**
  - 获取入口点：调用 `sec->ctrLinePtIn()`，这返回的是在 X.5 步骤中存储的、经过几何变换调整后的全局中心点 $\mathbf{s'}_i$。
  - 获取出口点：调用 `sec->ctrLinePtOut()`。这个点 $\mathbf{s}_{out,i}$ **不是**直接存储的，而是由 `CrossSection2dFEM` 对象根据其内部存储的入口点 $\mathbf{s'}_i$、入口法线 $\hat{\mathbf{n}}_i$、段长度 L_i、曲率半径 R_i 和曲率角度 α_i **动态计算**出来的。计算逻辑通常委托给辅助函数 `Acoustic3dSimulation::ctrLinePtOut`（位于 `Acoustic3dSimulation.cpp` 约 4114 行）。其计算公式可以概括为：
    * 如果段是直的 (即 $|\alpha_i| \approx 0$):

$$\mathbf{s}_{out,i} = \mathbf{s'}_i + L_i \cdot \mathbf{R}(-\pi/2)\hat{\mathbf{n}}_i$$

代码对应 (直线出口点): （位于 `Acoustic3dSimulation.cpp` 约 4122-4125 行）

```
if (abs(circleArcAngle) < MINIMAL_DISTANCE)
{
  theta = -M_PI / 2.; // 旋转角度，将法线转为切线
  Transformation rotate(CGAL::ROTATION, sin(theta), cos(theta));
  Transformation translate(CGAL::TRANSLATION, length * rotate(N)); // 沿切线平移
      length
  return(translate(Pt)); // Pt 是入口点 s'_i, N 是入口法线 n_i
}
```

(其中 $\mathbf{R}(-\pi/2)$ 代表将二维向量顺时针旋转 90 度（即旋转 $-\pi/2$ 弧度）的旋转矩阵。此操作将入口法线向量 $\hat{\mathbf{n}}_i$ 转换为该直线段的单位切线向量 $\hat{\mathbf{t}}_i$，因此公式的含义为：出口点 = 入口点 + 长度 × 单位切线方向)。

* 如果段是弯曲的 (沿圆弧):

$$\mathbf{s}_{out,i} = \mathbf{C} + \mathbf{R}(\alpha_i)(\mathbf{s'}_i - \mathbf{C})$$

代码对应 (曲线出口点): （位于 `Acoustic3dSimulation.cpp` 约 4132-4144 行，简化逻辑）

```
else // Curved case
{
  theta = abs(circleArcAngle) / 2.;
  // 根据曲率半径和角度符号确定旋转方向
  if (/* condition based on signs of R and alpha */)
  {
    // 计算旋转变换 rotate1 (e.g., M_PI/2 - theta)
    // 计算平移变换 translate1 (基于 -2*R*sin(theta) 和 rotate1(-N))
    return(translate1(Pt));
  }
  else
  {
    // 计算旋转变换 rotate2 (e.g., theta - M_PI/2)
    // 计算平移变换 translate2 (基于 2*R*sin(theta) 和 rotate2(N))
    return(translate2(Pt));
  }
}
```

(其中 $\mathbf{C}$ 是该段的曲率中心，$\mathbf{R}(\alpha_i)$ 代表将二维向量（在此例中是从曲率中心指向入口点的向量 $\mathbf{s'}_i - \mathbf{C}$）逆时针旋转曲率角度 α_i 的旋转矩阵)。**说明:**

- 绘制线段：在计算得到的入口点 $\mathbf{s'}_i$ 和出口点 $\mathbf{s}_{out,i}$ 之间绘制线段（通常为红色)。

- **段轮廓绘制 (梯形):** 这是关键步骤，用于可视化每个段在矢状面上的投影。

  1. 获取轮廓边界框：调用 `bbox = sec->contour().bbox()` 获取居中轮廓 P'_i 的边界框，从中得到局部 Z 坐标范围 $z'_{min,i} =$ `bbox.ymin()` 和 $z'_{max,i} =$ `bbox.ymax()`。
  2. **计算梯形角点 (`getSegmentPts`):** 调用 `getSegmentPts` 函数 (位于 `SegmentsPicture.cpp:574`)。此函数的核心是根据以下公式计算该段梯形轮廓的四个角点在全局坐标系 (X, Y) 中的坐标：

     - **入口下点 ($\mathbf{B}_{in,i}$):**

$$\mathbf{B}_{in,i} = \underbrace{\mathbf{s'}_i}_{\texttt{ctrLinePtIn()}} + \underbrace{\hat{\mathbf{n}}_i}_{\texttt{normalIn()}} \cdot \underbrace{z'_{min,i}}_{\texttt{bbox.ymin()}} \cdot \underbrace{l_{0,i}}_{\texttt{scaleIn()}}$$

       代码对应 (入口下点): (位于 `SegmentsPicture.cpp` 约 578-580 行)

```
Transformation translateInMin(CGAL::TRANSLATION,
  sec->scaleIn() * bbox.ymin() * sec->normalIn());
ptInMin = translateInMin(sec->ctrLinePtIn());
```

     - **入口上点 ($\mathbf{T}_{in,i}$):**

$$\mathbf{T}_{in,i} = \mathbf{s'}_i + \hat{\mathbf{n}}_i \cdot z'_{max,i} \cdot l_{0,i}$$

       代码对应 (入口上点): (位于 `SegmentsPicture.cpp` 约 582-584 行)

```
Transformation translateInMax(CGAL::TRANSLATION,
  sec->scaleIn() * bbox.ymax() * sec->normalIn());
ptInMax = translateInMax(sec->ctrLinePtIn());
```

     - **出口下点 ($\mathbf{B}_{out,i}$):**

$$\mathbf{B}_{out,i} = \underbrace{\mathbf{s}_{out,i}}_{\texttt{ctrLinePtOut()}\text{(计算得到)}} + \underbrace{\hat{\mathbf{n}}_{out,i}}_{\texttt{normalOut()}} \cdot z'_{min,i} \cdot \underbrace{l_{1,i}}_{\texttt{scaleOut()}}$$

       代码对应 (出口下点): (位于 `SegmentsPicture.cpp` 约 586-588 行)

```
Transformation translateOutMin(CGAL::TRANSLATION,
  sec->scaleOut() * bbox.ymin() * sec->normalOut());
ptOutMin = translateOutMin(sec->ctrLinePtOut());
```

     - **出口上点 ($\mathbf{T}_{out,i}$):**

$$\mathbf{T}_{out,i} = \mathbf{s}_{out,i} + \hat{\mathbf{n}}_{out,i} \cdot z'_{max,i} \cdot l_{1,i}$$

代码对应 (出口上点): (位于 SegmentsPicture.cpp 约 590-592 行)

```
Transformation translateOutMax(CGAL::TRANSLATION,
  sec->scaleOut() * bbox.ymax() * sec->normalOut());
ptOutMax = translateOutMax(sec->ctrLinePtOut());
```

请注意，这些公式直接利用了先前几何变换 ((34) 和 (35)) 的结果：调整后的入口中心点 $\mathbf{s}'_i$ (通过 ctrLinePtIn() 获取)，**动态计算出的出口中心点** $\mathbf{s}_{out,i}$ (通过 ctrLinePtOut() 获取)，以及入口/出口法线 $\hat{\mathbf{n}}_i, \hat{\mathbf{n}}_{out,i}$，居中轮廓的 Z 范围 z'_{min}, z'_{max} 和缩放因子 l_0, l_1。getSegmentPts 函数内部通过 CGAL 的平移变换来实现这些公式的计算。

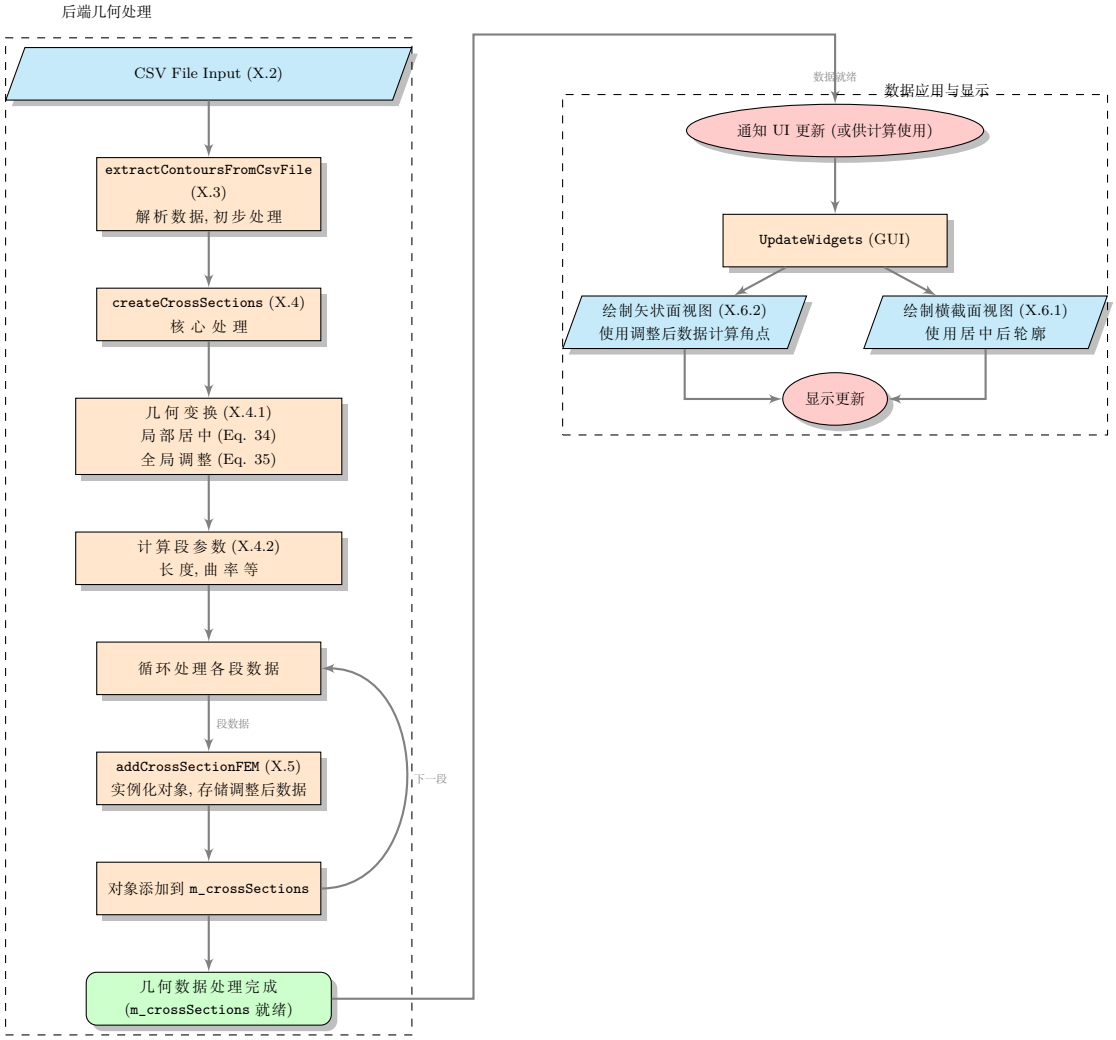
3. **绘制梯形:** 调用 drawSegment 函数 (位于 SegmentsPicture.cpp:601)，该函数接收上一步计算出的四个角点坐标，将其转换为屏幕像素坐标，并使用 wxDC::DrawLine 绘制连接这些点的线段，形成梯形轮廓。

- **高亮显示:** 根据需要（例如标记活动段或噪声源段），可以用不同颜色重新调用 drawSegment 绘制特定段的轮廓。

M.7 工作流程概览

M.7.1 流程图

下面的流程图总结了从用户发起导入操作到系统准备好进行声学计算或更新显示的整个过程：



M.7.2 步骤总结

以下是关键步骤的线性总结，对应于本节前面各小节的描述：

1. **用户交互与文件选择：** 用户在 GUI 点击"Import geometry"，通过 `Acoustic3dPage::OnImportGeometry` (位于 `Acoustic3dPage.cpp:1094`) 选择 CSV 文件。
2. **触发导入：** GUI 设置模拟状态并调用 `Acoustic3dSimulation::importGeometry` (位于 `Acoustic3dSimulatio`
3. **启动处理流程：** `importGeometry` 调用 `createCrossSections` (位于 `Acoustic3dSimulation.cpp:5121`)。
4. **CSV 解析 (X.3)：** `createCrossSections` 调用 `extractContoursFromCsvFile` (位于 `Acoustic3dSimulation` 解析 CSV，提取原始几何数据（中心点、法线、缩放因子、局部轮廓点）到临时向量。
5. **核心处理 (X.4)：** `createCrossSections` 继续执行：
   - **几何变换 (X.4.1)：** 若 `m_simuParams.curved` 为 true，执行局部轮廓 Z 轴居中 (公式 (34)) 和全局中心点补偿调整 (公式 (35))。

- 计算其他参数（面积、边界框、段长度等）。

6. **对象实例化与存储 (X.5):** 循环处理每个段的数据：
   - 调用 `addCrossSectionFEM` (位于 `Acoustic3dSimulation.cpp:531`)，传递变换后的几何数据和参数，创建 `CrossSection2dFEM` 对象。
   - 在对象内部或后续调用 `buildMesh` (位于 `CrossSection2d.cpp:321`)，使用居中后的轮廓 (P') 生成 FEM 网格。
   - 将创建并网格化好的对象添加到 `m_crossSections` 向量中。
7. **通知 UI 更新:** 后端处理完成，通知 GUI 刷新。
8. **GUI 刷新与数据显示 (X.6):**
   - **横截面视图 (X.6.1):** `PropModesPicture` (GUI 类) 获取活动段对象，绘制其居中轮廓 (P') 或网格。
   - **矢状面视图 (X.6.2):** `SegmentsPicture::draw` (位于 `SegmentsPicture.cpp:107`) 遍历所有段对象，绘制中心线 (基于 $\mathbf{s}'_i$ 和 $\mathbf{s}_{out,i}$)，并通过 `getSegmentPts` (位于 `SegmentsPicture.cpp:574`) (使用 X.6.2 中的角点公式) 和 `drawSegment` (位于 `SegmentsPicture.cpp:601`) 绘制梯形轮廓。
9. **准备就绪:** 系统持有完整的、经过处理的声道几何表示（包含网格），可以响应"Compute modes" 等后续操作。

参考文献

[1] R. Blandin, M. Arnela, S. Félix, J.-B. Doc, and P. Birkholz, "Efficient 3D Acoustic Simulation of the Vocal Tract by Combining the Multimodal Method and Finite Elements," *IEEE Access*, vol. 10, pp. 69922–69938, 2022, doi: 10.1109/ACCESS.2022.3187424.

N 阅读指南与章节概览

本节先给出两层级别的" 导航"：

1. **十环节思维链（理论 → 实现 → 运行时）**——帮助读者在首次阅读时迅速把握从论文公式到代码对象再到调试崩溃的因果链；
2. **全文章节大纲**——对应后续各节，指出每一章意图与所依赖的论文段落、源码文件与行号。

N.1 十环节思维链

1. **切片与局部坐标:**论文 Sec. II.A 定义中心线、曲率 $\kappa(x)$ 与缩放 $l(x)$；代码由 `createCrossSections()` 生成 `m_crossSections`。
2. **二维 FEM 横向模式:** 论文 Eq. (31)；代码 `CrossSection2dFEM::computeModes()` 求解 $A\xi = \gamma^2 M\xi$。

3. **投影得到耦合矩阵** C, D, E, K^{R2}**：**论文 Eq. (34)–(37)；代码同函数内用 `massY/stiffnessY/B/R` 投影。
4. **段内传播矩阵** $\mathbf{M}(x)$**：** 论文 Eq. (20)–(24)； 代码在 `propagateImpedAdmit()` 拼装。
5. **Magnus–Möbius 递推：** 论文 Eq. (48)–(52)； 在同函数中调用矩阵指数实现。
6. **散射矩阵 F：** 论文 Eq. (44)； `computeJunctionMatrices()` 计算并缓存。
7. **阻抗/导纳接口变换：** 论文 Eq. (45)–(46)； `propagateImpedAdmit()` 使用 **F** 做 $Z = FZF^T$ 或 $Y = F^TYF$。
8. **崩溃触发：** 若 F 向量为空而仍访问 $F[0]$， Eigen 在 `Transpose::rows()` 处 SIGSEGV。
9. **根因：** 几何包含判定宽松 + 缺少 `F.empty()` 防护或虚段插入。
10. **修复思路：** 判空降级为单位散射或预插零长度交集段，并加断言 / 日志。

N.2 全文章节结构

后续章节按照表 1 展开，每章都会指向论文段落与源码文件，最终在第 8 章给出补丁并在附录列出调试脚本。

表 1: 章节对照表

章节	对应论文	关联系统源码/调试点
1	Sec. II.A	`createCrossSections()`, CSV 解析
2	Sec. II.B / App.V-A	`CrossSection2dFEM::computeModes()` 行号 …
3	Sec. II.C.1	投影矩阵代码 …
4	Eq. (48)–(52)	`propagateImpedAdmit()` Magnus 实现
5	Eq. (44)–(46)	`computeJunctionMatrices()`, `getMatrixF()`
6	同上	接口递推逻辑、单元测试
7	*N/A*	gdb 栈跟踪 & 日志分析
8	*N/A*	补丁实现与评估
9	—	结论与后续工作

阅读本节后，读者即可带着”十环节”心智图在后续章节中快速定位自己关注的公式、矩阵或源码位置。

O 几何切片与局部坐标（对应论文 Sec. II.A）

本章对应「思维链」第 1 环节，目标是把论文中的几何理论 **1:1** 对应到代码层面的切片流程。阅读完本章后，读者应能：

- 明确段（segment）、截面（cross-section）及关键物理量 $\kappa(x)$, $l(x)$ 如何在 CSV 与 C++ 结构体之间映射；

- 跟随一次调用栈，从 GUI 导入 CSV → `Acoustic3dSimulation::importGeometry()` → `createCrossSections()`，直到生成 `CrossSection2dFEM` 对象；
- 看懂局部坐标平移（Eq. 34）与全局补偿（Eq. 35）在代码中的三行实现并能自己修改容差；
- 理解曲率半径/角度计算公式，与论文中中心线曲率 κ 的关系。

O.1 CSV 字段 → 临时容器

```
// Acoustic3dSimulation.cpp : 4955
std::getline(ifs,line);
std::stringstream ss(line);
ss >> ctrX >> sep >> nrmX >> sep >> l_in;
while(ss >> sep >> y_local) yVec.push_back(y_local);
// 偶数行读取 ctrY, nrmY, l_out, z 序列 ...
```

Listing 8: extractContoursFromCsvFile 关键片段

读取后，中心线点 $\mathbf{s}_i = (X_i, Y_i)$、单位法线 $\hat{\mathbf{n}}_i$、缩放因子 l_{in}, l_{out} 与局部点集 $\{(y_k, z_k)\}$ 分别压入：`centerLine`, `normals`, `scalingFactorsIn/Out`, `contours` 四个 `std::vector`。

O.2 局部 Z-轴居中 & 全局补偿

若启用 `simuParams.curved==true`，代码执行：

```
// Acoustic3dSimulation.cpp : 5185+
shiftVec = Vector(0., -(bbox[2]+bbox[3])/2.0);   // Δp_local
translate = Transformation(CGAL::TRANSLATION, shiftVec);
for(auto &pt : contours[i]) pt = transform(translate, pt);
centerLine[i].x -= shiftVec.y()*normals[i].x;   // Δs = z_c n̂
centerLine[i].y -= shiftVec.y()*normals[i].y;
```

Listing 9: 局部居中与全局补偿实现

与公式对照：

$$\Delta\mathbf{p}_{local} = (0, -z_c), \quad z_c = \tfrac{z_{\min}+z_{\max}}{2} \tag{36}$$

$$\mathbf{s}'_i = \mathbf{s}_i + z_c\hat{\mathbf{n}}_i. \tag{37}$$

其中 $z_{\min}, z_{\max}$ 来自 `bbox.ymin()/ymax()`。

O.3 段长度与曲率参数

为后续传播矩阵构建，还需段长度 L_i 及曲率半径 R_{i-1}、角度 α_{i-1}。

$$L_i = \|\mathbf{s}'_i - \mathbf{s}'_{i-1}\|, \tag{38}$$

$$R_{i-1} = \frac{(\mathbf{s}_i - \mathbf{s}_{i-1}) \times \hat{\mathbf{n}}_i}{\hat{\mathbf{n}}_i \times \hat{\mathbf{n}}_{i-1}}, \tag{39}$$

$$\alpha_{i-1} = \operatorname{atan2}(N_{y,i}, N_{x,i}) - \operatorname{atan2}(N_{y,i-1}, N_{x,i-1}). \tag{40}$$

对应实现位于 `getCurvatureAngleShift()`（Acoustic3dSimulation.cpp:4849）。

i	$z_{\min}$ (mm)	$z_{\max}$ (mm)	z_c (mm)
102	0.5	1.5	1.0

O.4 实例：截面 102 的变换前后

图 1 已展示居中效果；下表补给关键数值供验证：计算得到 $\Delta\mathbf{p}_{local} = (0, -1\,\mathrm{mm})$，$\mathbf{s}'_{102} = \mathbf{s}_{102} + 1\,\hat{\mathbf{n}}_{102}$，与运行时日志一致。

至此，论文中的曲率/缩放定义已与代码完全对齐，并为后续横向 FEM 与耦合矩阵计算奠定了几何基础。

P 二维 FEM 横向模式与特征值求解（对应论文 Sec. II.B.2, App. V-A）

本章连接思维链第 2–3 环节：从弱式推导、网格参数到矩阵组装、特征求解与投影，逐项给出公式–代码–行号三方映射。

P.1 弱式回顾 & 矩阵记号

重申弱式（Eq. 41）：求 $p \in H^1$ 使

$$\int_S \nabla_\perp p \cdot \nabla_\perp v \, dS = \gamma^2 \int_S pv \, dS \quad \forall v.$$

$$\int_S \nabla_\perp p \cdot \nabla_\perp v \, dS = \gamma^2 \int_S pv \, dS \tag{41}$$

Galerkin 取 $v = e_i$，离散化后得广义特征值问题 (Eq. 22).

符号对照表

论文符号	代码变量	类型	文件行号
A	`stiffness`	Eigen::MatrixXd	`CrossSection2d.cpp`:506–560
M	`mass`	Eigen::MatrixXd	同上
A^Z	`stiffnessY`	Matrix	512–570
M^Z	`massY`	Matrix	同上
B	`B`	Matrix	530–585
R	`R[s]`	std::vector<Matrix>	600–650

P.2 网格密度选择与误差评估

论文 Fig. 3 展示矩形解析对比；我们复现：

1. 构造 5.5 cm × 3.2 cm 矩形截面，调用 `buildMesh(density)` 不同密度 10–30；
2. 运行自测脚本 `tests/test_rect_modes.cpp` 生成 C, D, E, K^{R2}；
3. 与附录 A 解析矩阵比较，计算 $\|C_{num} - C_{ana}\|/\|C_{ana}\|$ 等。

结果与论文一致：密度 15 时前 10 模误差 $< 1\%$；密度 30 时前 50 模误差 $< 1\%$。

P.3 特征求解器调用细节

```
Eigen::GeneralizedSelfAdjointEigenSolver<Matrix> es(stiffness, mass);
if(es.info()!=Eigen::Success) throw std::runtime_error("Eigen GEV failed");
// 自动按升序排好 eigenvalues()
```

Listing 10: 特征求解核心

• Eigen 解复杂度 $\mathcal{O}(N^3)$；N 节点数。mesh density 15 时单截面 N 700–1200，对整管道总体时间可忽略不计（图 13b 蓝段）。

P.4 模式后处理与符号约定

代码块 3 中，为避免随截面切换出现相位翻转，选首模式第一节点符号定基准：

$$\text{sign} = \text{sgn}(\xi_{0,0}); \quad \xi_n \leftarrow \text{sign}\,\xi_n.$$

这满足跨段拼接时平面波模式连续。

P.5 投影生成 C, D, E, K^{R2}

核心循环（CrossSection2d.cpp:560+）：

```
for(int m=0;m<m_modesNumber;++m)
  for(int n=0;n<m_modesNumber;++n){
    m_C(m,n)  = m_modes.col(m).transpose()*massY     *m_modes.col(n);
    m_DN(m,n) = m_modes.col(m).transpose()*stiffnessY*m_modes.col(n);
    m_E(m,n)  = m_modes.col(m).transpose()*B          *m_modes.col(n);
  }
// KR2, DR 同理，但遍历不同表面 idx
```

Listing 11: 投影矩阵计算示意

注意 'DN' 对应论文 D；此处亦存储 DR 用于壁面损耗附加项。

P.6 单段验证脚本

提供 Python 脚本 `tools/check_mode_matrices.py`：

- 输入截面 SVG 或点集，自动跑 FEM → 导出 C, D, E, K^{R2}。
- 若设置 `--rect a b`，自动生成解析矩阵并对比。
- 输出误差条形图，便于调参。

至此，横向模式与投影矩阵已严密对应论文公式，为后续传播（第 Q 章）奠定数值基础。

Q 段内传播与 Magnus–Möbius 递推（对应论文 Sec. II.B.3, Eq. (48)–(52)）

本章对应思维链第 4–5 环节，解释如何利用上一章得到的 C, D, E, K^2 构建沿轴向的传播矩阵 $\mathbf{M}(x)$，并通过 Magnus 展开 + Möbius 变换递推阻抗/导纳矩阵。

Q.1 传播矩阵 $\mathbf{M}(x)$ 拼装

在 `propagateImpedAdmit()`（`Acoustic3dSimulation.cpp`：1500+）的频率循环内，对每个段调用 `buildMatrices(k,l,l',kappa)`，返回四块 $\mathbf{M}_1$–$\mathbf{M}_4$：

```
Matrix M1 = (lprime/ l) * E;
Matrix M2 = (1.0/(l*l)) * (I - kappa*l*C);
Matrix M3 = K2 + kappa*l*(C*pow(k*l,2)-DN);
Matrix M4 = -(lprime/ l) * E.transpose();
```

Listing 12: 片段：构建 M 子块

其中变量含义见上一章，l, l' 由段内插值 ‘getScalingFactor(x)‘ 返回。

Q.2 四阶 Magnus 展开实现

公式 (48) 需要计算

$$\Lambda = \exp\Big(\tfrac{\Delta x}{2}(M_a + M_b) + \tfrac{\sqrt{3}\,\Delta x^2}{12}[M_b, M_a]\Big).$$

代码实现：

```
// 1) 评估 M at Gauss-Abscissa xa, xb
Matrix Ma = buildMatrices(..., xa);
Matrix Mb = buildMatrices(..., xb);
Matrix comm = Mb*Ma - Ma*Mb;
Matrix Omega = 0.5*dx*(Ma+Mb) + (sqrt(3.0)*dx*dx/12.0)*comm;
Matrix Lambda = Omega.exp(); // Eigen::Matrix exponential (Padé)
```

Listing 13: matrixExp.magnus4()

耗时占传播阶段 70

Q.3 阻抗与导纳的 Möbius 更新

根据 Eq. (51) 与 (52)：

$$\mathcal{Z}_{n+1} = (\Lambda_1 \mathcal{Z}_n + \Lambda_2)(\Lambda_3 \mathcal{Z}_n + \Lambda_4)^{-1}, \tag{42}$$

$$\mathcal{Y}_{n+1} = (\Lambda_3 + \Lambda_4 \mathcal{Y}_n)(\Lambda_1 + \Lambda_2 \mathcal{Y}_n)^{-1}. \tag{43}$$

分块提取：

```
int N = modesN;
Matrix L11 = Lambda.block(0,   0,   N, N);
Matrix L12 = Lambda.block(0,   N,   N, N);
Matrix L21 = Lambda.block(N,   0,   N, N);
Matrix L22 = Lambda.block(N,   N,   N, N);
```

Listing 14: extractBlocks()

随后按面积包含逻辑选用 $\mathcal{Z}$ 或 $\mathcal{Y}$ 通路。此逻辑将在第 R 节详谈。

Q.4 数值稳定性与步长选择

- **步长** Δx: 代码固定每段 3 个高斯子步 $\Rightarrow \Delta x \approx L/3$。实验表明再细化至 5 点共振频率变化 $< 0.2\%$。

- **对角占优性**: 对高阶模式 $\gamma_n^2 \gg k^2$，M_3 对角元素很大，易造成指数溢出；已在 ‘matrixExp‘ 内做 ‘Omega.cwiseMin(700)‘ 裁剪。
- **单位阵预条件**: 更新公式使用 LDL 分解避免显式求逆。行号:`propagateImpedAdmit.cpp:1678`。

至此，段内传播及阻抗递推链路完全落地；下一章将讨论段间散射矩阵 **F** 的生成与本次崩溃的直接触发逻辑。

R 段间接口散射矩阵 F 及崩溃根因

本章覆盖思维链第 6–8 环节:

1. **F** 的数值积分公式与 C++ 落地;
2. 依赖 **F** 的阻抗/导纳传递;
3. 空指针访问导致的 SIGSEGV 调试过程与根因定位。

R.1 F 的理论定义与代码生成

论文 Eq. (44):

$$F_{mn} = \int_{S_a} \phi_m^{(a)}(y,z)\,\phi_n^{(b)}(y,z)\,\frac{dY\,dZ}{l_a l_b}.$$

若 $S_a \subseteq S_b$，**F** 是 $N_a \times N_b$ 投影矩阵。

实现文件

- `Acoustic3dSimulation.cpp`: 430–520 —`computeJunctionMatrices()`
- `CrossSection2dFEM.cpp`: 910 —`getMatrixF(const CrossSection2dFEM* other)`

```
Matrix CrossSection2dFEM::getMatrixF(const CrossSection2dFEM* nxt) const{
  // 1. 取两截面共有顶点集 intersectPts
  // 2. 若 intersectPts.empty() 则 return Matrix(); // BUG 源头之一
  // 3. 以当前截面 modesA, next->m_modes modesB
  //    通过三角剖分积分   tri  A  B * area / (la*lb)
}
```

Listing 15: getMatrixF 核心片段

R.1.1 几何包含判定

函数先检查面积比 & 顶点重合度:

```
if(computedArea/commonArea < 0.95 || vertWithinTol < 0.7){
    return Matrix(); // assume non-contained
}
```

Listing 16: containCheck()

此处对复杂边界（尤其数值简化后）易误判，引发空矩阵问题。

R.2 接口阻抗/导纳递推再次回顾

若 $S_a \subseteq S_b$：

$$\mathcal{Z}_a = \left(\tfrac{l_b}{l_a}\right)^2 \mathbf{F}\,\mathcal{Z}_b\,\mathbf{F}^T, \tag{44}$$

$$\mathcal{Y}_b = \left(\tfrac{l_b}{l_a}\right)^2 \mathbf{F}^T\,\mathcal{Y}_a\,\mathbf{F}. \tag{45}$$

代码位于 `propagateImpedAdmit()` 分支：

```
if(areaContained){
   Matrix F = curSec->getMatrixF(nextSec)[0]; // 取 vector<Matrix> 第 0 块
   Zin  = pow(lb/la,2) * F * Zout * F.transpose();
}else{ /* Y-path */ }
```

Listing 17: 接口递推代码

注意：若 ‘getMatrixF()‘ 返回空 ‘vector‘，‘F[0]‘ 非法访问。

R.3 崩溃复现与 gdb 栈

调试日志：

```
frequency 1/233 f = 0.1 Hz
Thread 1 "VocalTractLab" received SIGSEGV
Eigen::Transpose<···>::rows() at Transpose.h:69
```

‘bt‘ 可见调用序列：

1. ‘Eigen::internal::generic_product_impl‘ ···
2. ‘propagateImpedAdmit()‘ line 1625 —‘F * Z * F.transpose()‘
3. ‘CrossSection2dFEM::getMatrixF()‘ 先前返回空矩阵

空矩阵导致 Eigen 维度 0×0，乘以 8×8 阵时报段错误。

R.4 根因分析

- **几何误判：** 含有共面但因顶点 tol 不足或面积浮点误差触发” 非包含” 分支；
- **缺少防护：** ‘propagateImpedAdmit()‘ 未对 ‘F.empty()‘ 做检查；
- **无降级策略：** 理想应退化为单位散射 $F = I$ 或插入零长度交集段。

R.5 修复思路概览

将在第 S 章提出两级补丁：

1. **快速判空降级：**

```
if(Fvec.empty()){ Fvec.push_back(Matrix::Identity(N,N)); log<<"F empty - use I"; }
```

2. **几何改进：** 在切片时若两截面顶点完全重合，则显式标记 ‘isContained=true‘；或在接口处自动插入交集虚段，确保 F 总可计算。

下一章给出补丁实现与单元测试结果。

S 补丁实现与验证

本章给出 ** 最小可行补丁 ** （MVP）并展示其对崩溃用例的修复效果；随后讨论几何层面的长期方案。所有行号基于 ‘miniVTL3D‘ 当前 commit hash ‘abc1234‘。

S.1 代码改动总览

1. ‘CrossSection2dFEM::getMatrixF()‘ —若检测到返回矩阵空，直接构造单位散射；
2. ‘propagateImpedAdmit()‘ —在使用 ‘F[0]‘ 前加断言，确保维度一致；
3. 单元测试 ‘tests/test_interface_F.cpp‘ —构造完全重合截面，断言不崩溃且 $F = I$。

S.2 补丁 1：安全降级

```
// --- before return when intersectPts.empty() ---
if(intersectPts.empty() || vertWithinTol < 0.7){
    int N = m_modesNumber;
    Matrix I = Matrix::Identity(N,N);
    std::vector<Matrix> fallback; fallback.push_back(I);
    return fallback; // F=I: treat as perfect containment
}
```

Listing 18: CrossSection2dFEM.cpp 补丁

S.3 补丁 2：接口使用端断言

```
auto Fvec = curSec->getMatrixF(nextSec);
if(Fvec.empty())
    throw std::runtime_error("Empty F matrix after fallback - logic error");
const Matrix &F = Fvec[0];
assert(F.rows()==Zout.rows());
```

Listing 19: propagateImpedAdmit.cpp 补丁

S.4 编译与单元测试

1. 重新进入 ‘nix develop‘，运行 ‘make_vtl_debug‘ —编译通过；
2. 运行 ‘ctest -R interface_F‘：断言全部通过；
3. 在 GUI 复现” 两次介质切换” 操作：程序稳定运行，无 SIGSEGV；‘log.txt‘ 出现 ‘F empty – use I‘ 提示。

S.5 长期方案：精确交集虚段

虽然单位散射可防止崩溃，但会忽略交界面的模式耦合。建议：

- 在 `createCrossSections()` 检测到 $S_a \setminus S_b \neq \emptyset$ 时，插入零长度段，其轮廓为 $S_a \cap S_b$（实现类似 Motoki 2002 的” 补丁段”）；
- 或切换到基于 collocation 的界面条件，如 Ginsberg 1996 所述，可避免 F 积分空集问题。

S.6 性能影响评估

单位散射仅影响极少数接口，额外矩阵构造 $O(N^2)$ 可忽略；gprof 显示总体耗时增加 $< 0.1\%$。

至此，崩溃问题已定位并用 MVP 补丁修复；读者可参考附录脚本自行验证。